Orientação a Objetos
-
Programação em C++

2º Slides: Relações entre objetos em C++

Prof. Jean Marcelo SIMÃO

Retomando a última aula

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;

public:
  // no .h há a assinatura dos métodos ou funções-membro
  Pessoa ( int diaNa, int mesNa, int anoNa );
  void Calc_Idade ( int diaAT, int mesAT, int anoAT );
  int informaIdade ( );    // int getIdade();
};
```

Pessoa.cpp

```
#include "Pessoa.h“
 // no .cpp há a implementação dos métodos funções-membro
Pessoa::Pessoa ( int diaNa, int mesNa, int anoNa )
{
   ...
}

void Pessoa::Calc_Idade ( int diaAT, int mesAT, int anoAT )
{
  . . .
 }

int Pessoa::informaIdade ( )          // int getIdade()
{
   return idadeP;
}
```

main.cpp

```
#include "Pessoa.h"
int main()
{
   Pessoa Einstein ( 14, 3, 1879 );
   Pessoa Newton ( 4, 1, 1643 );

   Einstein.Calc_Idade ( 25, 8, 2009 );
   Newton.Calc_Idade ( 25, 8, 2009 );

   printf ( " Einstein teria %d \n", Einstein.informaIdade ( ) );
   printf ( " Newton teria %d   \n", Newton.informaIdade ( ) );

   getchar();
   return 0;
}
```

O encapsulamento

Exercício proposto anteriormente: fazer com que a “impressão” (*printf...*) da idade ocorra dentro dos próprios objetos, programando isto no método apropriado da classe Pessoa...

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
    Pessoa Einstein ( 14, 3, 1879 );
    Pessoa Newton (    4, 1, 1643 );

    Einstein.Calc_Idade ( 25, 8, 2009 );
    Newton.Calc_Idade ( 25, 8, 2009 );

    printf("Einstein teria %d \n", Einstein.informaIdade());
    printf("Newton teria %d   \n", Newton.informaIdade());

    getchar();
    return 0;
}
```

Resolvendo o exercício – Atributo NomeP

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP [30];

public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char nome[] );
   void Calc_Idade ( int diaAT, int mesAT, int anoAT );
   int informaIdade ( );
};
```

main.cpp

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
  Pessoa Simao    (  3, 10, 1976, "Jean Simao" );
  Pessoa Einstein ( 14, 3, 1879, "Albert Einstein" );
  Pessoa Newton   ( 4, 1, 1643, "Isaac Newton" );

  Simao.Calc_Idade    ( 25, 08, 2009);
  Einstein.Calc_Idade ( 25, 08, 2009);
  Newton.Calc_Idade   ( 25, 08, 2009);

  getchar();
  return 0;
}
```

Pessoa.cpp

```
#include "Pessoa.h“
#include <string.h>
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char nome[])
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
    idadeP = 0;
}

void Pessoa::Calc_Idade (int diaAT, int mesAT, int anoAT)
{
     idadeP = anoAT - anoP;
     if (mesP < mesAT)
     {
        idadeP = idadeP - 1;
     }  else
     {
        if (mesP == mesAT)
        {
           if (diaP < diaAT)
           {
              idadeP = idadeP - 1;
           }
        }
     }
    // o comando passa a ser encapsulado dentro do método...
     printf("A idade da Pessoa %s seria %d \n", nomeP, idadeP);
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

Resolvendo o exercício – valor por *default*

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP [30];

public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char nome[] =
"");
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```

main.cpp

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
  Pessoa Simao    (  3, 10, 1976 );
  Pessoa Einstein ( 14, 3, 1879, "Albert Einstein");
  Pessoa Newton   (  4, 1, 1643, "Isaac Newton");

  Simao.Calc_Idade    ( 25, 08, 2009 );
  Einstein.Calc_Idade ( 25, 08, 2009 );
  Newton.Calc_Idade   ( 25, 08,  2009 );

  getchar();
  return 0;
}
```

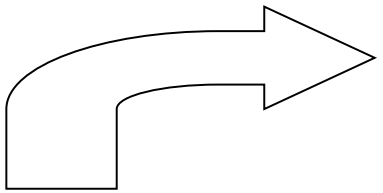
Pessoa.cpp

```
#include "Pessoa.h“
#include <string.h>
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char nome[])
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
  ....
}

void Pessoa::Calc_Idade (int diaAT, int mesAT, int anoAT)
{
    idadeP = anoAT - anoP;
    if (mesP < mesAT)
    {
       idadeP = idadeP - 1;
    }
    else {
       if (mesP == mesAT)
       {
          if (diaP < diaAT)
          {
             idadeP = idadeP - 1;
          }
       }
    }
   // o comando passa a ser encapsulado dentro do método...
    printf("A idade da Pessoa %s seria %d \n", nomeP, idadeP);
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

Resolvendo o exercício – valor por *default*

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP [30];

public:
   Pessoa(int diaNa, int mesNa, int anoNa = 0, char nome[] = "");
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```



main.cpp

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
  Pessoa Simao    (3, 10 );
  Pessoa Einstein (14, 3, 1879, "Albert Einstein");
  Pessoa Newton  (4, 1, 1643, "Isaac Newton");

  Simao.Calc_Idade    (25, 02, 2007);
  Einstein.Calc_Idade (25, 02, 2007);
  Newton.Calc_Idade  (25, 02,  2007);

  getchar();
  return 0;
}
```

Na verdade, qualquer atributo pode ter um valor por *default* (por 'defeito' ou padrão), desde que o(s) parâmetro(s) mais a direita tenha(m cada qual) um valor por default também.

Assim sendo, o primeiro exemplo que segue é valido enquanto o segundo não.

Certo:

```
Pessoa ( int diaNa = 0,
         int mesNa = 0,
         int anoNa = 0,
         char nome[] = "");
```

Errado:

```
Pessoa ( int diaNa = 0,
         int mesNa = 0,
         int anoNa,
         char nome[] = "");
```

Obs.: Os valores defaut dos parâmetros aparecem na assinatura da função-menbro (no .h) mas não aparecem na implementação dela (no .cpp).

Resolvendo o exercício – valor por *default*

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP [30];

public:
   Pessoa(int diaNa, int mesNa, int anoNa, char nome[] = "");
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```

main.cpp

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
  Pessoa Simao    (3, 10, 1976, "Jean Simao");
  Pessoa Einstein (14, 3, 1879, "Albert Einstein");
  Pessoa Newton  (4, 1, 1643, "Isaac Newton");

  Simao.Calc_Idade    (25, 02, 2007);
  Einstein.Calc_Idade (25, 02, 2007);
  Newton.Calc_Idade  (25, 02,  2007);

  getchar();
  return 0;
}
```

Pessoa.cpp

```
#include "Pessoa.h“
#include <string.h>
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char nome[])
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
   idadeP = 0;
}

void Pessoa::Calc_Idade (int diaAT, int mesAT, int anoAT)
{
    idadeP = anoAT - anoP;
    if (mesP < mesAT)
    {
       idadeP = idadeP - 1;
    } else
    {
       if (mesP == mesAT)
       {
          if (diaP < diaAT)
          {
             idadeP = idadeP - 1;
          }
       }
    }
   // o comando passa a ser encapsulado dentro do método...
    printf ("A idade da Pessoa %s seria %d \n", nomeP, idadeP);
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

Resolvendo o exercício – ponteiro para caracter.

Pessoa.h

```
#include <stdio.h>

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP [30];

public:
   Pessoa(int diaNa, int mesNa, int anoNa, char* nome);
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```

main.cpp

```
#include <stdio.h>
#include "Pessoa.h"
int main()
{
  Pessoa Simao    (3, 10, 1976, "Jean Simao");
  Pessoa Einstein (14, 3, 1879, "Albert Einstein");
  Pessoa Newton   (4, 1, 1643, "Isaac Newton");

  Simao.Calc_Idade    (25, 02, 2007);
  Einstein.Calc_Idade (25, 02, 2007);
  Newton.Calc_Idade   (25, 02,  2007);

  getchar();
  return 0;
}
```

Pessoa.cpp

```
#include "Pessoa.h“
#include <string.h>
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
   idadeP = 0;
}

void Pessoa::Calc_Idade (int diaAT, int mesAT, int anoAT)
{
     idadeP = anoAT - anoP;
     if (mesP < mesAT)
     {
        idadeP = idadeP - 1;
     }  else
     {
        if (mesP == mesAT)
        {
           if (diaP < diaAT)
           {
              idadeP = idadeP - 1;
           }
        }
     }
    // o comando passa a ser encapsulado dentro do método...
     printf("A idade da Pessoa %s seria %d \n", nomeP, idadeP);
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

Trocando *printf* por *cout*.

- Em C++, usa-se normalmente *cout* em vez de printf.

```
printf ( " A idade da  Pessoa  %s  seria  %d   \n  ",  nomeP,  idadeP );

cout << "A idade da Pessoa " << nomeP << " seria" << idadeP << endl;
```

- O *cout* é um comando que trabalha orientado a fluxo...

Aumentando o encapsulamento

Pessoa.h

```
class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];

public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char* nome = "" );
   void Calc_Idade ( int diaAT, int mesAT, int anoAT );
   int informaIdade ( );
};
```

main.cpp

```
#include "Pessoa.h“

int main()
{
  Pessoa Simao (  3, 10, 1976, "Jean Simao" );
  Pessoa Einstein ( 14, 3, 1879, "Albert Einstein" );
  Pessoa Newton (  4, 1, 1643, "Isaac Newton" );

  Simao.Calc_Idade ( 25, 08, 2009 );
  Einstein.Calc_Idade ( 25, 08, 2009 );
  Newton.Calc_Idade ( 25, 08,  2009 );
  getchar();
  return 0;
}
```

Pessoa.cpp

```
. . .
Pessoa::Pessoa ( int diaNa, int mesNa, int anoNa, char* nome )
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
}

void Pessoa::Calc_Idade ( int diaAT, int mesAT, int anoAT )
{
     idadeP = anoAT - anoP;
     if (mesP < mesAT)
     {
        idadeP = idadeP - 1;
     }
     else
     {
        if (mesP == mesAT)
        {  if (diaP < diaAT)
           {
              idadeP = idadeP - 1;
           }
        }
     }
    // o comando passa a ser encapsulado dentro do método...
     cout << "A idade da Pessoa " << nomeP << " seria"
          << idadeP << endl;
}

int Pessoa::informaIdade ( )
 {
     return idadeP;
}
```

Resolvendo o exercício (com *cout*)

Pessoa.h

```
class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];

public:
   Pessoa(int diaNa, int mesNa, int anoNa, char* nome =
"");
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```

main.cpp

```
#include "Pessoa.h“

int main()
{
  Pessoa Simao(3, 10, 1976, "Jean Simao");
  Pessoa Einstein(14, 3, 1879, "Albert Einstein");
  Pessoa Newton(4, 1, 1643, "Isaac Newton");




  getchar();
  return 0;
}
```

Pessoa.cpp

```
. . .
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
   Calc_Idade (25, 08, 2009);
}

void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT)
{
    idadeP = anoAT - anoP;
    if (mesP < mesAT)
    {
       idadeP = idadeP - 1;
    }
    else
    {
       if (mesP == mesAT)
       {  if (diaP < diaAT)
          {
             idadeP = idadeP - 1;
          }
       }
    }
   // o comando passa a ser encapsulado dentro do método...
    cout << "A idade da Pessoa " << nomeP << " seria"
         << idadeP << endl;
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

Resolvendo o exercício (com *cout*)

Pessoa.h

```
class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];

public:
   Pessoa(int diaNa, int mesNa, int anoNa, char* nome =
"");
   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   int informaIdade();
};
```

main.cpp

```
// #include <iostream.h>
#include <iostream>
using namespace std;
#include "Pessoa.h"
int main()
{
  Pessoa Simao(3, 10, 1976, "Jean Simao");
  Pessoa Einstein(14, 3, 1879, "Albert Einstein");
  Pessoa Newton(4, 1, 1643, "Isaac Newton");


  cout << " Fim do Programa. "  << endl

  getchar();
  return 0;
}
```

Pessoa.cpp

```
. . .
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy(nomeP, nome);
   Calc_Idade ( 25, 08, 2009 );
}

void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT)
{
    idadeP = anoAT - anoP;
    if (mesP < mesAT)
    {
       idadeP = idadeP - 1;
    }
    else
    {
       if (mesP == mesAT)
       {  if (diaP < diaAT)
          {
             idadeP = idadeP - 1;
          }
       }
    }
   // o comando passa a ser encapsulado dentro do método...
    cout << "A idade da Pessoa " << nomeP << " seria"
         << idadeP << endl;
}

int Pessoa::informaIdade()
 {
 return idadeP;
}
```

```
#include "Pessoa.h“
#include <string.h>
#include <iostream>
using std::cout;
using std::endl;

Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy ( nomeP,  nome );
}

void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT)
{
    idadeP = anoAT - anoP;
    if (mesP > mesAT)
    {
       idadeP = idadeP - 1;
    }
    else
    {
       if (mesP == mesAT)
       {  if (diaP > diaAT)
          {
             idadeP = idadeP - 1;
          }
       }
    }
   // o comando passa a ser encapsulado dentro do método...
    cout << "A idade da Pessoa " << nomeP << "  seria"  << idadeP << endl;
}

int Pessoa::informaIdade()
{
    return idadeP;
}
```

Usa-se

```
#include <iostream>
using std::cout;
using std::endl;
...
```

ou

```
#include <iostream.h>
```

Isto depende de compilador, sendo a primeira a forma mais “moderna”.

Fluxo de entrada – *cin* e *cout*.

```
#include <stdio.h>
#include "Pessoa.h“
...

int main()
{

   int dia, mes, ano;

  Pessoa Simao (3, 10, 1976, "Jean Simao");
  Pessoa Einstein (14, 3, 1879, "Albert Einstein");
  Pessoa Newton (4, 1, 1643, "Isaac Newton");

  cout << " Informe o dia atual: " << endl;
  cin >> dia;

  cout << " Informe o mês atual: " << endl;
  cin >> mes;

  cout << " Informe o ano atual: " << endl;
  cin >> ano;

  Simao.Calc_Idade (dia, mes, ano);
  Einstein.Calc_Idade (dia, mes, ano);
  Newton.Calc_Idade (dia, mes, ano);

  getchar();
  return 0;
}
```

O cin e o cout tratam de entradas e saidas por meio de FLUXO

O fluxo de entrada é cin >> .

O fluxo de saída é cout << .

Reflexão

Quando se programa em C++, deve-se fazê-lo o mais orientado a objetos possível...

Neste sentido, em orientação a objetos, o programa deveria começar já a partir de um objeto de uma classe tida como principal.

Entretanto, os exemplos mostrados até então começam a partir de uma função, a função *main*. Na verdade, o C++ nos obriga a começar a partir da função *main*.

Assim sendo, como resolver este impasse, esta "deficiência" do C++?

Exercício 1.

- Criar uma classe Principal, onde cada objeto que venha existir no sistema (e.g. Objetos da Classe Pessoa) sejam criados ou instanciados.

- A função *main()* deverá instanciar somente um objeto da classe principal, respeitando assim uma boa conduta de elaboração de sistemas orientados a objetos.

Obs.: Para tratar entrada e saída utilizar *cin* e *cout*.

Modificando a classe Pessoa

```
class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];
public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char* nome = "" );

   Pessoa ();

   void Inicializa ( int diaNa, int mesNa, int anoNa, char* nome = "" );
   void Calc_Idade ( int diaAT, int mesAT, int anoAT );
   int informaIdade ();
};
```

Toda classe deve ter uma função-membro construtora sem parâmetros ...

```
#include "Pessoa.h"
Pessoa::Pessoa ( int diaNa, int mesNa, int anoNa, char* nome )
{
    Inicializa (diaNa, mesNa, anoNa, nome);
}

Pessoa::Pessoa()
 {
   Inicializa (0, 0, 0);

 }

void Pessoa::Inicializa( int diaNa, int mesNa, int anoNa, char* nome )
{
   idadeP = 0;
   diaP = diaNa;
   mesP = mesNa;
   anoP = anoNa;
   strcpy ( nomeP, nome );
}
. . .
```

Objeto Principal – V1

Principal.h

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

  public:
    Principal();
    void Executar();
};
```

Principal.cpp

```
#include "Principal.h"
Principal:: Principal ()
{
    Simao.Inicializa     (3, 10, 1976, "Jean Simão");
    Einstein.Inicializa  (14, 3, 1879, "Albert Einstein");
    Newton.Inicializa    (4, 1, 1643, "Isaac Newton");
}

void Principal::Executar()
{
    Simao.Calc_Idade  (25, 8, 2009);
    Einstein.Calc_Idade (25, 8, 2009);
    Newton.Calc_Idade  (25, 8, 2009);
}
```

```
#include "Principal.h"
int main()
{
  // Declaração do Objeto Principal
  Principal objetoPrincipal;
  // Execução do Objeto Principal
  objetoPrincipal.Executar();
  getchar();
  return 0;
}
```

Objeto Principal – V1

Principal.h

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

  public:
    Principal();
    void Executar();
};
```

Principal.cpp

```
#include "Principal.h"
Principal:: Principal ()
{
    Simao.Inicializa     (  3, 10, 1976, "Jean Simão" );
    Einstein.Inicializa  ( 14, 3, 1879, "Albert Einstein" );
    Newton.Inicializa    (  4, 1, 1643, "Isaac Newton" );
    Executar ();
}

void Principal::Executar()
{
    Simao.Calc_Idade    ( 25, 8, 2009 );
    Einstein.Calc_Idade ( 25, 8, 2009 );
    Newton.Calc_Idade   (25, 8, 2009 );
}
```

Principal.h **Principal.cpp**

```
#include "Principal.h"
int main()
{
  // Declaração do Objeto Principal
  Principal objetoPrincipal;

  getchar();
  return 0;
}
```

main.cpp

Objeto Principal – V2

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

Principal.h

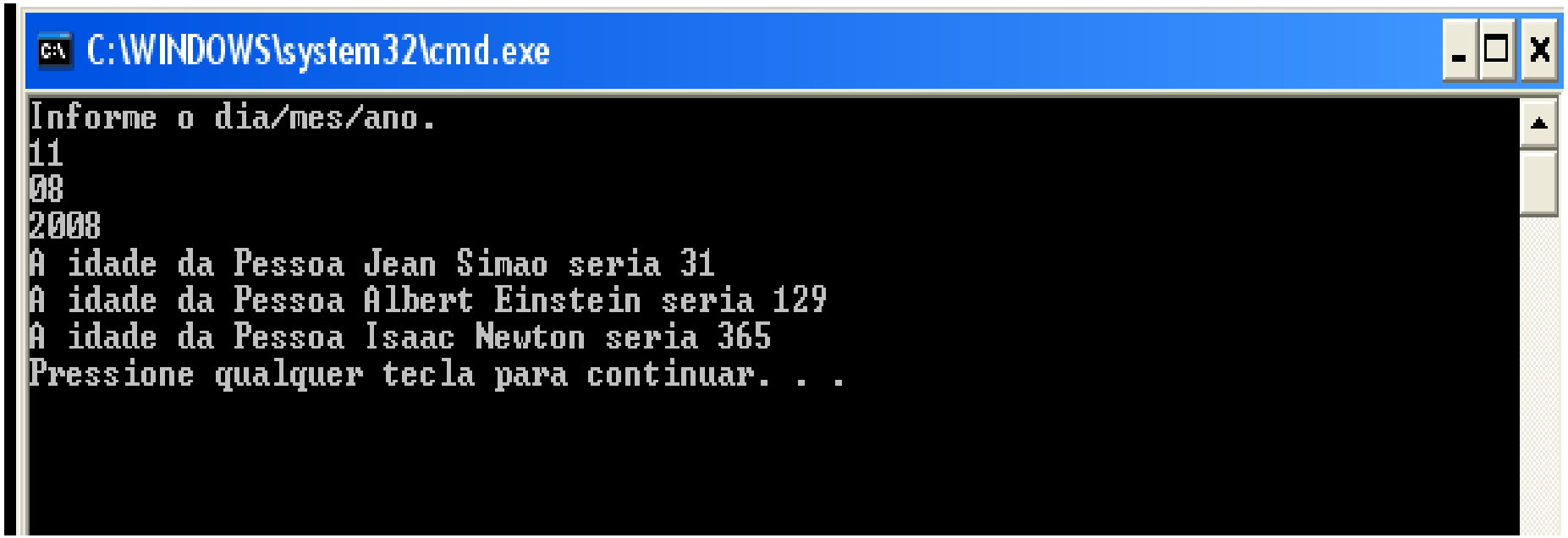
```
#include "Principal.h“
...
Principal::Principal()
{
    Simao.Inicializa    ( 3, 10, 1976, "Jean Simão" );
    Einstein.Inicializa ( 14, 3, 1879, "Albert Einstein" );
    Newton.Inicializa   ( 4, 1, 1643, "Isaac Newton" );

    cout << " Informe o dia/mes/ano." << endl;
    cin >> diaAtual >> mesAtual >> anoAtual;

    Executar();
}

void Principal::Executar();
{
    // Execução
    Simao.Calc_Idade ( diaAtual, mesAtual, anoAtua );
    Einstein.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    Newton.Calc_Idade ( diaAtual, mesAtual, anoAtual );
}
```

Principal.cpp

```
#include "Principal.h"
...
int main()
{
    // Declaração do Objeto Principal
    // Principal ObjetoPrincipal ();
    Principal ObjetoPrincipal;

    getchar();
    return 0;
}
```

main.cpp

Execução do Programa



Objeto Principal – V3 (time.h)

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

```
#include "Principal.h“
#include <time.h>
Principal::Principal()
{
    Simao.Inicializa    ( 3, 10, 1976, "Jean Simão" );
    Einstein.Inicializa ( 14, 3, 1879, "Albert Einstein" );
    Newton.Inicializa   ( 4, 1, 1643, "Isaac Newton" );

    // O ponteiro 'local' é do tipo struct tm, que contém data e hora.
     struct tm *local;
     time_t segundos;
     time(&segundos);
     local = localtime(&segundos);

     // Retorna o dia.
     diaAtual = local->tm_mday;

     // Retorna o mês.
     mesAtual = local->tm_mon + 1;

    // Retorna o ano.
    // É necessário acrescentar 1900, pois o sistema retorna a partir desse ano.
    // Ao invés de 2009, retorna 109.
    anoAtual = local->tm_year + 1900;

    Executar() ;
}

void Principal::Executar()
{
    Simao.Calc_Idade        ( diaAtual, mesAtual, anoAtua );
    Einstein.Calc_Idade     ( diaAtual, mesAtual, anoAtual );
    Newton.Calc_Idade       ( diaAtual, mesAtual, anoAtual );
}
```

Há um certo *bug* (falha) do time.h previsto para 2038.

Vide http://code.google.com:80/p/y2038/

(Página acessada em 13/02/2009)

Obs.: Esta versão foi elaborada pela aluna Thayse S. em fevereiro de 2009.

Objeto Principal – V4 (Windows.h)

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

```
#include "Principal.h“
#include <time.h>
#include <Windows.h>

Principal::Principal()
{
    Simao.Inicializa    ( 3, 10, 1976, "Jean Simão" );
    Einstein.Inicializa ( 14, 3, 1879, "Albert Einstein" );
    Newton.Inicializa   ( 4, 1, 1643, "Isaac Newton" );

    SYSTEMTIME st;
    GetSystemTime ( &st );

     diaAtual = st.wDay;
     mesAtual = st.wMonth;
     anoAtual = st.wYear;

    Executar() ;
}

void Principal::Executar()
{

     Simao.Calc_Idade         ( diaAtual, mesAtual, anoAtua );

    Einstein.Calc_Idade       ( diaAtual, mesAtual, anoAtual );

    Newton.Calc_Idade         ( diaAtual, mesAtual, anoAtual );

}
```

Relação entre Objetos

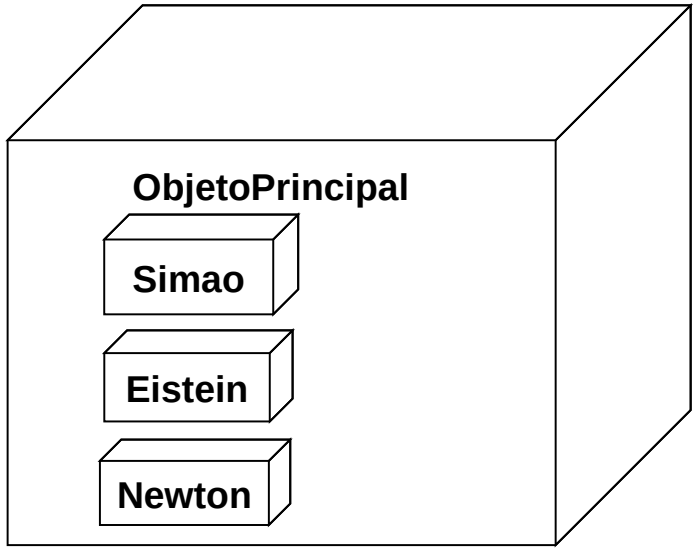
Agregação-forte

Relação entre Objetos – Agregação-forte.

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

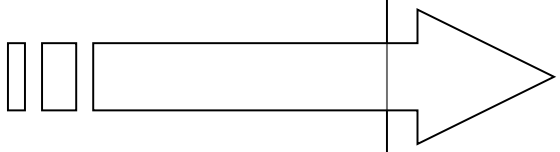
A classe Principal define que seu(s) objeto(s) deve(m) ter um relacionamento com três objetos provenientes (instanciados) da classe Pessoa.

Ao bem da verdade, estes três objetos estão incluídos ou agregados no (“ou em cada”) objeto da classe Principal.



Chamada de Construtora sem parâmetros.

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```



Neste código em questão, quando um objeto da classe Principal é criado, são criados três objetos da classe Pessoa, nomeadamente Simao, Einstein e Newton.

Segundo este código, quando cada objeto da Classe Pessoa é criado, é automaticamente chamado seu construtor sem parâmetro.

```
#include "Principal.h"
Principal::Principal()
{
   Simao.Inicializa(3, 10, 1976, "Jean Simão");
   Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
   Newton.Inicializa(4, 1, 1643, "Isaac Newton");
}

void Principal::Executar()
{
   . . .
   Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
   Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
   Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

```
#include "Principal.h“
...
int main()
{
   // Declaração do Objeto Principal
   Principal ObjetoPrincipal;
   // Execução do Objeto Principal
   ObjetoPrincipal.Executar ();

   getchar();
   return 0;
}
```

Construção dos Objetos

```
#include "Principal.h“
...
int main()
{
   // Declaração do Objeto Principal
   Principal ObjetoPrincipal;
   // Execução do Objeto Principal
   ObjetoPrincipal.Executar ();

   getchar();
   return 0;
}
```

- Cria o objeto *ObjetoPrincipal* da classe Principal.
  - Cria o objeto *Simao* da Classe Pessoa.
    - Chama a Construtora (sem parâmetros) de Simao
  - Cria o objeto *Einstein* da Classe Pessoa.
    - Chama a Construtora (sem parâmetros) de Einstein.
  - Cria o objeto *Newton* da Classe Pessoa.
    - Chama a Construtora (sem parâmetros) de Newton
- Chama a construtora do *ObjetoPrincipal*

Chamada de Construtora com parâmetros.

```
#include "Pessoa.h"
class Principal
{
  private:

    Pessoa Simao ( 3, 10, 1976, "Jean Simão“ );

    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

ERRO, ISTO NÃO FUNCIONA!

```
#include "Principal.h"
Principal::Principal()
{

    // Simao.Inicializa (3, 10, 1976, "Jean Simão");
    Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
    Newton.Inicializa(4, 1, 1643, "Isaac Newton");
}

void Principal::Executar()
{
    . . .
    Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
    Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
    Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

Chamada de Construtora com parâmetros.

```
#include "Pessoa.h"
class Principal
{
  private:

    Pessoa Simao;

    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

ISTO SIM FUNCIONA!

```
#include "Principal.h“

Principal::Principal ( ) :
Simao ( 3, 10, 1976, “Jean Simão”)
{

    // Simao.Inicializa (3, 10, 1976, "Jean Simão");
    Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
    Newton.Inicializa(4, 1, 1643, "Isaac Newton");
}

void Principal::Executar()
{
    . . .
    Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
    Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
    Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

Chamada de Construtora com parâmetros.

```
#include "Pessoa.h"
class Principal
{
  private:

    Pessoa Simao;

    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:
    Principal();
    void Executar();
};
```

ISTO SIM FUNCIONA!

```
#include "Principal.h“

Principal::Principal ( ) :
Simao ( 3, 10, 1976, “Jean Simão”),
Einstein ( 14, 3, 1879, “Albert Einstein”),
Newton ( 4, 1, 1643, “Isaac Newton”)
{
    // Simao.Inicializa (3, 10, 1976, "Jean Simão");
    // Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
    // Newton.Inicializa(4, 1, 1643, "Isaac Newton");
    . . .

}

void Principal::Executar()
{
    Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
    Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
    Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

Chamada de Construtora sem parâmetros explicitamente

```
#include "Pessoa.h"
class Principal
{
  private:

    Pessoa Simao;

    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:

    Principal ();
   void Executar();

};
```

```
#include "Principal.h“

Principal::Principal ( ) :
Simao ( ) ,
Einstein ( ) ,
Newton ( )
{

    Simao.Inicializa (3, 10, 1976, "Jean Simão");
    Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
    Newton.Inicializa(4, 1, 1643, "Isaac Newton");

     . . .

}

void Principal::Executar()
{
Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
    Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
    Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

Método ou Função-Membro *Destrutora*

```
#include "Pessoa.h"
class Principal
{
  private:

    Pessoa Simao;

    Pessoa Einstein;
    Pessoa Newton;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:

    Principal ();

    ~Principal ();

    void Executar();
};
```

```
#include "Principal.h“
Principal::Principal ( ) :

Simao ( ) ,
Einstein ( ) ,
Newton ( )

{
   Simao.Inicializa (3, 10, 1976, "Jean Simão");
   Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
   Newton.Inicializa(4, 1, 1643, "Isaac Newton");
}

Principal:: ~Principal ( )

{
}

void Principal::Executar()
{
   . . .
   Simao.Calc_Idade(diaAtual, mesAtual, anoAtua);
   Einstein.Calc_Idade(diaAtual, mesAtual, anoAtual);
   Newton.Calc_Idade(diaAtual, mesAtual, anoAtual);
}
```

- A construtora inicializa elementos, como atributos. Ela também serve para alocar memória quando for o caso.
- A destrutora “zera” alguns elementos... Ela serve por exemplo para desalocar memória quando alguma memória tenha sido alocada em alguma função-membro do objeto...

- No exemplo em questão a destrutora é vazia e serve para atender a regra geral que toda classe deve ter uma destrutora vazia.
- Na verdade, toda a classe deve ter uma construtora e uma destrutora vazia para impedir que o compilador (em alguns casos) as crie automaticamente, fazendo com que o programador/desenvolvedor não controle isso...

Observação Técnica

- Os códigos apresentados acima foram elaborados e executados em projetos usando o Dev C++.

- Caso o aluno deseje refazer estes códigos em projeto em Microsoft Visual C++ Express Edition, haverão mais dois arquivos e alguns detalhes...

  – Veja os próximos slides...

A mesmo código em Visual C++ .net Express Edition

```
class Pessoa
{
private:
    int diaP;
    int mesP;
    int anoP;
    int idadeP;
    char nomeP[30];

public:
    Pessoa(int diaNa, int mesNa, int anoNa, char* nome = "");
    Pessoa();
    void Inicializa(int diaNa, int mesNa, int anoNa, char* nome = "");
    void Calc_Idade(int diaAT, int mesAT, int anoAT);
    int informaIdade();
};
```

```
#include "stdafx.h"
#include "Pessoa.h"

Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
  Inicializa(diaNa, mesNa, anoNa, nome);
}

. . .
```

```
#include "Pessoa.h"
class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

  public:
    Principal();
    void Executar();
};
```

```
#include "stdafx.h"
#include "Principal.h"
Principal::Principal()
{
    Simao.Inicializa(3, 10, 1976, "Jean Simao");
    Einstein.Inicializa(14, 3, 1879, "Albert Einstein");
    Newton.Inicializa(4, 1, 1643, "Isaac Newton");
}

void Principal::Executar()
{
    int dia, mes, ano;
    cout << "Informe o dia/mes/ano." << endl;
    cin >> dia >> mes >> ano;
    Simao.Calc_Idade(dia, mes, ano);
    Einstein.Calc_Idade(dia,mes,ano);
    Newton.Calc_Idade(dia, mes, ano);
}
```

```
// ProjetoOOExemplo.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "Principal.h"

int _tmain(int argc, _TCHAR* argv[])
{

  // declaração do Objeto Principal
  Principal ObjetoPrincipal;
  // Execução do Objeto Principal
  ObjetoPrincipal.Executar();

  return 0;
}
```

ProjetoOOExemplo.cpp

stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#define WIN32_LEAN_AND_MEAN          // Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here
//#include <stdio.h>
#include <string.h>

#include <iostream>
using std::cout;
using std::endl;
using std::cin;
```

stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// ProjetoOOExemplo.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

Exercícios

Conceitos por meio de exercícios.

Exercício 2 - Associação

- Criar uma classe chamada Universidade que terá como atributo um nome.

- Relacionar ou **associar** a classe Pessoa para com a classe Universidade. Cada objeto da classe Pessoa poderá ser **associado** a um objeto da classe Universidade.

  – A classe Pessoa terá um ponteiro para um objeto ou "variável" da classe Universidade.

- A classe Pessoa, por sua vez, terá uma função-membro (i.e. um método) que permitirá a cada objeto (de Pessoa) informar seu nome e em que universidade trabalha...

Classe Universidade

Universidade.h

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

class Universidade
{
  private:
     char nome[30];

  public:
     // Construtor
     Universidade ( char* n = “” );

     // Destrutor
     ~Universidade ( );

     void setNome ( char* n );
     char* getNome ( );
};

#endif
```

Universidade.cpp

```
#include "Universidade.h"
#include <string.h>

Universidade::Universidade ( char *n )
{
    strcpy ( nome, n );
}

Universidade::~Universidade ( )
{
}

void Universidade::setNome ( char* n )
{
   strcpy ( nome, n );
}

char* Universidade::getNome ( )
{
   return nome;
}
```

Obs. : O construtor inicializa elementos, como atributos.
O destrutor “zera” alguns elementos...

Observação – Cabeçalho de pré-compilação.

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

class Universidade
{
  private:
     char nome[30];

  public:
     Universidade();
     ~Universidade();

     void setNome ( char* n  = “” );
     char* getNome ( );
};

#endif
```

Cabeçalho de Pré-processador/Pré-compilação
ou
Diretivas de Pré-processador/Pré-compilação

Serve inclusive para tratar “includes repetidos...”

Universidade.h

Observação – Cabeçalho Simplificado

```
#pragma once


class Universidade
{
  private:
     char nome[30];

  public:
     Universidade();
     ~Universidade();

     void setNome ( char* n  = “” );
     char* getNome ( );
};
```

Faz o mesmo que o código anterior e é mais simples

Universidade.h

Ex.2: Associação entre os Objetos

```
#ifndef _PESSOA_H_
#define _PESSOA_H_

#include “Universidade.h”
. . .
class Pessoa
{
private:
   int diaP;  int mesP;  int anoP;  int idadeP;
   char nomeP[30];

   // pUnivFiliado é apenas uma referência a um objeto associado.
   Universidade* pUnivFiliado;
public:
   Pessoa (int diaNa, int mesNa, int anoNa, char* nome = "");
   Pessoa ( );
   void Inicializa (int diaNa, int mesNa, int anoNa, char* nome = "");
   void Calc_Idade (int diaAT, int mesAT, int anoAT);
   int informaIdade ();

   // Este método abaixo permite associar uma Univ. à Pessoa.
   void setUnivFiliado (Universidade* pu);
   void OndeTrabalho ();
};
#endif
```

```
#include "Pessoa.h"
. . .
void Pessoa::setUnivFiliado (Universidade* pu)
{
    pUnivFiliado = pu;
}

void Pessoa::OndeTrabalho ()
{
    // Um método da referência UnivFiliado é chamado.
    cout <<  nomeP <<" trabalha para a " << pUnivFiliado->getNome() << endl;
}
```

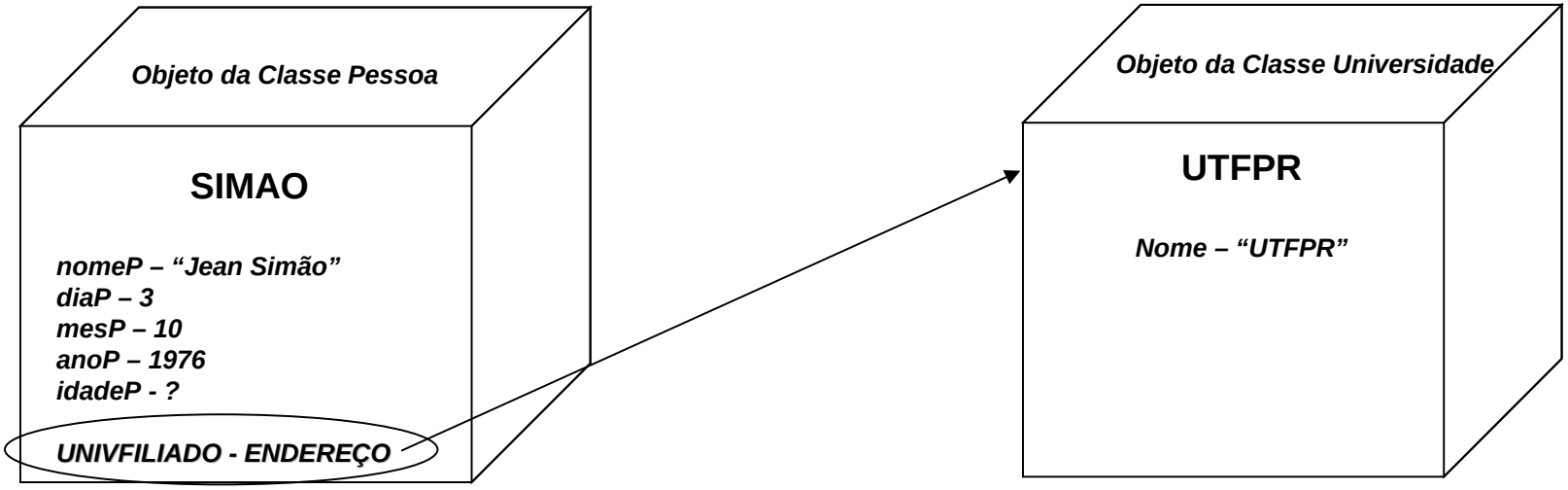
Ex.2: Associação entre Objetos

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Pessoa.h"
#include "Universidade.h“

class Principal
{
  private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    // UTFPR é agregada ao(s) objeto(s) desta classe!!!
    Universidade UTFPR;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:

    Principal();
    void Executar();
};

#endif
```

```
#include "Principal.h"
Principal::Principal()
{
    Simao.Inicializa ( 3, 10, 1976, "Jean Simão");
    Einstein.Inicializa ( 14, 3, 1879, "Albert Einstein");
    Newton.Inicializa ( 4, 1, 1643, "Isaac Newton");

    UTFPR.setNome ("UTFPR");

    // Aqui os objetos UTFPR e Simao são associados.
    // Na verdade, UTFPR é associado ao Simão via uma
    // passagem por referência do ‘endereço’ dela.
    Simao.setUnivFiliado(&UTFPR);

    Executar ( );
}

void Principal::Executar()
{
     . . .

    Simao.Calc_Idade ( diaAtual, mesAtual, anoAtua);
    Einstein.Calc_Idade (diaAtual, mesAtual,
anoAtual);
    Newton.Calc_Idade (diaAtual, mesAtual, anoAtual);

    Simao.OndeTrabalho();
}
```

Associação entre Objetos

```
#include "Principal.h"
Principal::Principal()
{
   Simao.Inicializa ( 3, 10, 1976, "Jean Simão" );
   Einstein.Inicializa ( 14, 3, 1879, "Albert Einstein" );
   Newton.Inicializa ( 4, 1, 1643, "Isaac Newton" );

   UTFPR.setNome ("UTFPR");

   Simao.setUnivFiliado ( &UTFPR );
}
```



Exercício 3 - Associação

- Criar dois objetos de Universidade associando um para Einstein e outro para Newton.
  - Einstein trabalhou como professor de física em Princeton (Nova Jersey - Estados Unidos da América).
  - Newton trabalhou como professor de matemática em Cambridge (Inglaterra).

Exercício 4 – Agregação e Associação

- Criar uma classe Departamento que permita agregar um objeto (Departamento) na classe Universidade.

- A classe Pessoa deve possuir uma referência ao departamento que trabalha, ou seja:

  – ela deve possuir uma associação com a Classe Departamento, permitindo que cada objeto Pessoa tenha a referência de um objeto Departamento.