OO – Engenharia Eletrônica

Orientação a Objetos
-
Programação em C++

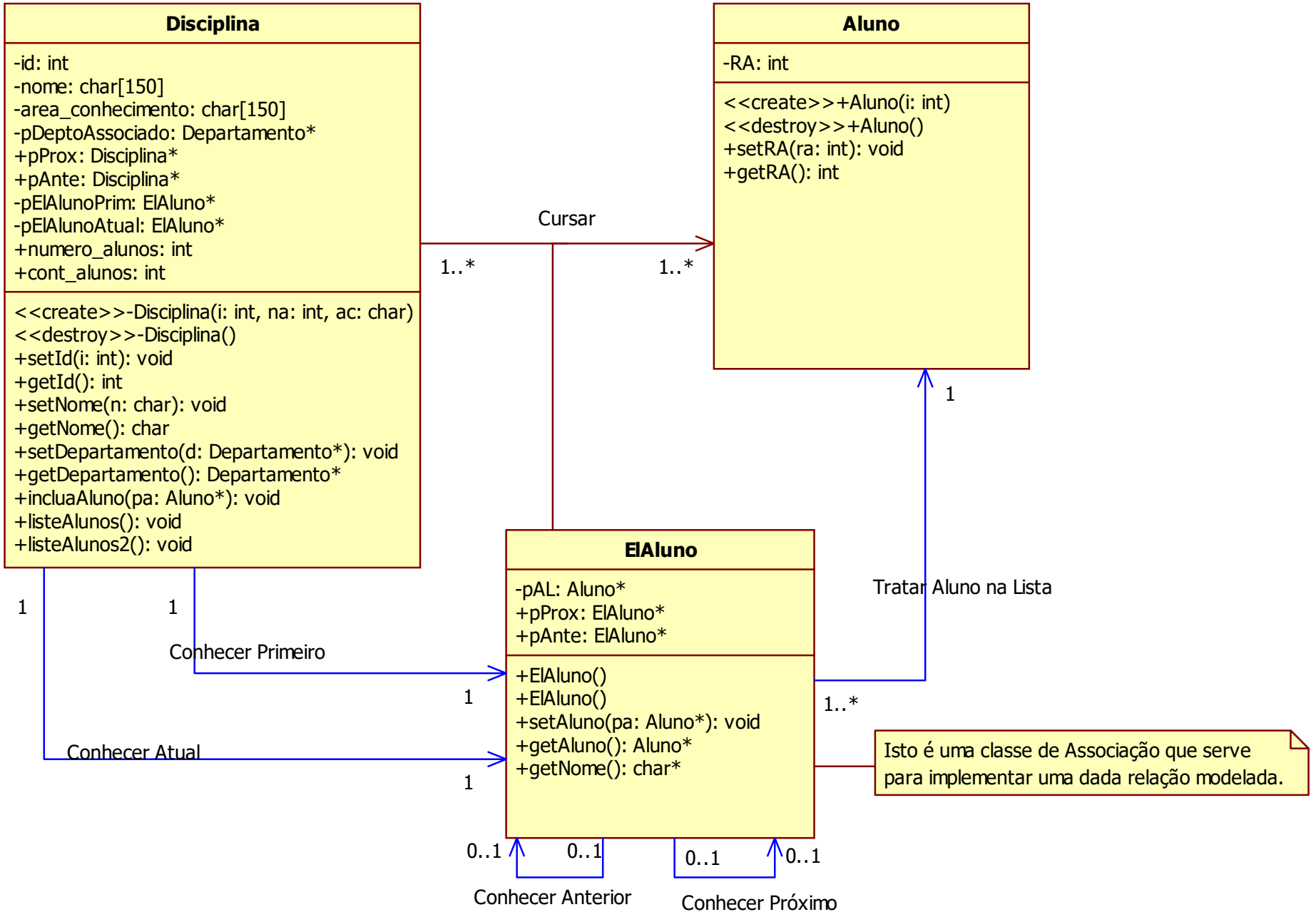
Slides 6: Listas Efetivamente e Alocação Dinâmica

Prof. Dr. Jean Marcelo SIMÃO – DAELN / UTFPR

Retomando últimos Slides

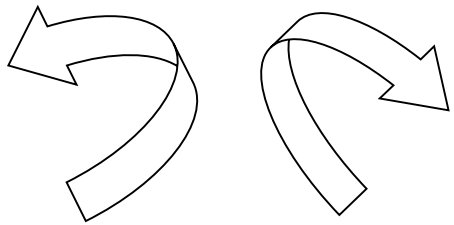
“Objetos ElAluno”

Diagrama de Classes – Projeto – Classe de Associação



```
#ifndef _LALUNO_H_
#define _LALUNO_H_
#include "Aluno.h"
class ElAluno
{
 private:
    Aluno* pAluno;
 public:
    ElAluno ( );
    ~ElAluno ( );
    ElAluno *pProx;
    ElAluno *pAnte;
    void setAluno ( Aluno* pa );
    Aluno* getAluno ( );
    char* getNome ( );
};
#endif
```

```
#include "LAluno.h"
#include <stdio.h>
// ...

void ElAluno::setAluno ( Aluno *pa )
{
   pAluno = pa;
}

Aluno* ElAluno::getAluno ( )
{
   return pAluno;
}

char* ElAluno::getNome ( )
{
    return pAluno->getNome ( );
}
```



A lista não será tratada na Classe *Aluno*, mas sim em uma outra classe relacionada.

```
#ifndef _ALUNO_H_
#define _ALUNO_H_
#include “Pessoa.h"

class Departamento;

class Aluno : public Pessoa
{

  private:
    int RA;
    Departamento* pDeptoAssociado;

  public:

    Aluno ( );
    ~Aluno ( );

    void setRA ( int ra );
    int getRA ( );

    void setDepartamento ( Departamento* pd );
    Departamento* getDepartamento ( );

};
#endif
```

```
#ifndef _DISCIPLINA_H_
#define _DISCIPLINA_H_
#include “ElAluno.h“
#include “Departamento.h”

class Disciplina
{
private:

   int id;
   char nome [ 150 ];
   char area_conhecimento [ 150 ];
   int numero_alunos;
   int cont_alunos;

   Departamento* pDeptoAssociado;

   ElAluno *pElAlunoPrim;
   ElAluno *pElAlunoAtual;

 public:

   Disciplina ( int na = 45, char* ac = "“ );
   ~Disciplina ( );

    ...

   void incluaAluno ( Aluno* pa );

   void listeAlunos ( );

   void listeAlunos2 ( );

};

#endif
```

```
void Disciplina::incluaAluno ( Aluno* pa )
{
   if ( ( cont_alunos < numero_alunos ) && ( pa != NULL ) )
   {
    // Aqui é criado um ponteiro para LAluno
    ElAluno* paux = NULL;
    // Aqui é criado um objeto LAluno, sendo seu endereço armazenado em aux
    paux = new ElAluno ( );
    // Aqui recebe uma cópia do objeto interm.
    paux->setAluno ( pa );

    if ( pElAlunoPrim == NULL )
    {
      pElAlunoPrim          = paux;
      pElAlunoAtual         = paux;
    }
    else
    {
     pElAlunoAtual->pProx   = paux;
     paux->pAnte            = pElAlunoAtual;
     pElAlunoAtual          = pAux;
    }
    cont_alunos++;
  }
  else
  {
     if ( pa != NULL )
     {
        printf ( " Aluno não incluído. Turma já lotada em %i alunos \n ", numero_alunos );
         // cout << . . ..
     }
     else
     {
        printf ( " Ponteiro nulo \n " );
     }
  }
}
```

ALOCAÇÃO DINÂMICA

Obs.: Não esquecer de substituir cada *printf* por um *cout*.

```
void Disciplina::listeAlunos()
{
   ElAluno* paux;
   paux = pElAlunoPrim;

   while ( paux != NULL )
   {
       printf ( " Aluno %s matriculado na Disciplina %s \n ", paux->getNome(), nome );
       paux = paux->pProx;
   }
}


void Disciplina::listeAlunos2 ( )
{
   ElAluno* paux;
   paux = pElAlunoAtual;

   while ( paux != NULL)
   {
      printf ( " Aluno %s matriculado na Disciplina %s \n", paux->getNome(), nome);
      paux = paux->pAnte;
   }

}
```

```
void Principal::ListeAlunosDisc ( )
{
   Metodos2_2007.listeAlunos ( );
   printf("\n");
   Metodos2_2007.listeAlunos2 ( );
   printf("\n");

   Computacao2_2007.listeAlunos ( );
   printf("\n");
   Computacao2_2007.listeAlunos2 ( );
   printf("\n");
}
```

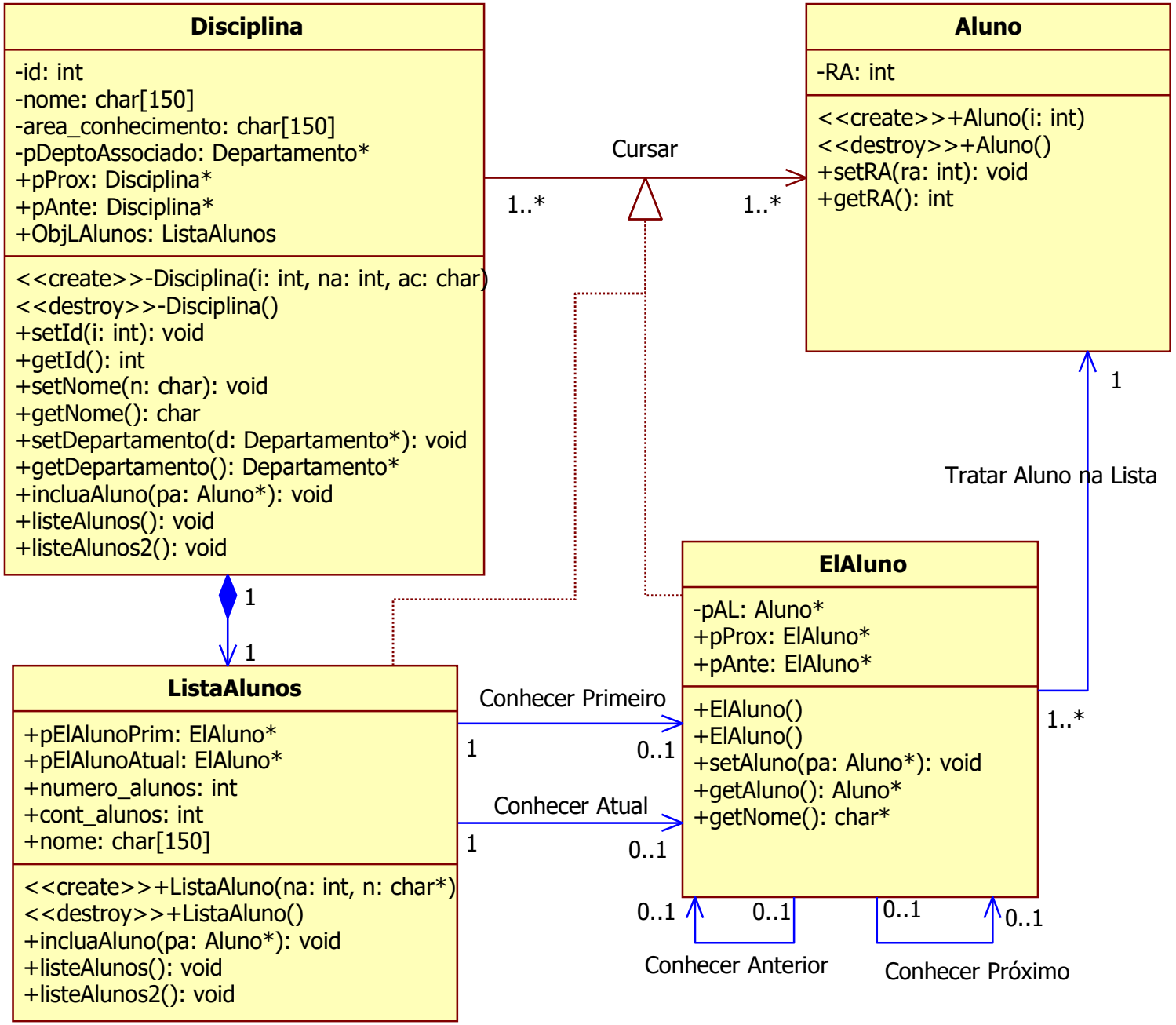
```
void Principal::Executar ( )
{
   CalcIdadeProfs ( );
   UnivOndeProfsTrabalham ( );
   DepOndeProfsTrabalham ( );
   ConhecPessoa ( );
   ListeDiscDeptos ( );
   ListeAlunosDisc ( );

   AAA.setNome ( "Teste" );
   printf ( "O novo nome de AAA é: %s \n", AAA.getNome ( ) );
   Computacao2_2007.listeAlunos ( );
}
```

Reflexão

```
#ifndef _DISCIPLINA_H_
#define _DISCIPLINA_H_
#include “ElAluno.h“
#include “Departamento.h”
class Disciplina
{
private:
    int         id;
    char        nome [ 150 ];
    char        area_conhecimento [ 150 ];
    int         numero_alunos;
    int         cont_alunos;
    Departamento* pDeptoAssociado;

  ElAluno *pElAlunoPrim;
  ElAluno *pElAlunoAtual;

  public:

    Disciplina ( int na = 45, char* ac = "“ );
    ~Disciplina ( );

     ...
    void incluaAluno ( Aluno* pa );
    void listeAlunos ( );
    void listeAlunos2 ( );
};
#endif
```

Isto lhes parece ‘filosoficamente’ correto?

Melhorando a Solução



Melhorando a Solução

```
#ifndef _LISTAALUNOS_H_
#define _LISTAALUNOS_H_

#include "ElAluno.h"
#include "Aluno.h"

class ListaAlunos
{
private:

   int         cont_alunos;
   int         numero_alunos;
   char        nome [ 150 ];

   ElAluno*    pElAlunoPrim;
   ElAluno*    pElAlunoAtual;

public:

   ListaAlunos ( int na, char* n );
   ~ListaAlunos ( );

   void incluaAluno ( Aluno* pa );
   void listeAlunos ( );
   void listeAlunos2 ( );

};

#endif
```

```
#ifndef _DISCIPLINA_H_
#define _DISCIPLINA_H_

#include "ListaAlunos.h"
#include "Departamento.h"

class Disciplina
{
private:
   int         id;
   char        nome [ 150 ] ;
   char        area_conhecimento [ 150 ];
   Departamento* pDeptoAssociado;

   ListaAlunos ObjLAlunos;

 public:
   Disciplina ( int na = 45, char* ac = "" );
   ~Disciplina ( );
   Disciplina* pProx;
   Disciplina* pAnte;

   void setId  ( int i );
   int getId ( );
   void setNome ( char* n );
   char* getNome ( );

   void setDepartamento ( Departamento* pd );
   Departamento* getDepartamento ( );

   void incluaAluno ( Aluno* pa );
   void listeAlunos ( );
   void listeAlunos2 ( );
};

#endif
```

```
#include "stdafx.h"
#include "ListaAlunos.h“

ListaAlunos::ListaAlunos ( int na, char* n )
{
   numero_alunos = na;
   cont_alunos      = 0;

   pElAlunoPrim  = NULL;
   pElAlunoAtual = NULL;

   strcpy  ( nome, n );
}

ListaAlunos::~ListaAlunos ( )
{
  ElAluno *paux1, *paux2;

  paux1 = pElAlunoPrim;

  paux2 = paux1;

  while ( paux1 != NULL )
  {
       paux2 = paux1->pProx;
       delete ( paux1 );
       paux1 = paux2;
  }

  pElAlunoPrim  = NULL;
  pElAlunoAtual = NULL;

}
```

```
void ListaAlunos::incluaAluno ( Aluno* pa )
{
   if ( ( cont_alunos < numero_alunos ) && ( pa != NULL) )
   {
     ElAluno* paux = NULL;
     paux = new ElAluno ( );
     paux->setAluno ( pa );
     if ( pElAlunoPrim == NULL )
     {
       pElAlunoPrim   = paux;
       pElAlunoAtual  = paux;
     }
     else
     {
       pElAlunoAtual->pProx    = paux;
       paux->pAnte             = pElAlunoAtual;
       pElAlunoAtual           = paux;
     }
     cont_alunos++;
   }
   else
   {
      if  ( pa != NULL )
      {
          cout << "Aluno não incluído. Turma já lotada em "
                 << numero_alunos << " alunos." << endl;
      }
      else
      {
          cout << " Ponteiro do Aluno está nulo! " << endl;
      }
   }
}
```

```
void ListaAlunos::listeAlunos ( )
{
   // paux é um ponteiro de (objeto da classe) pElAluno
   ElAluno* paux = NULL;

   // paux recebe o mesmo endereço de pElAlunoPrim, que é outro ponteiro (de objeto da classe) pELAluno.
   paux = pElAlunoPrim;

   while (aux != NULL)
   {
      // printf(" Aluno %s matriculado na Disciplina %s. \n", aux->getNome(), nome);

       cout << " Aluno " << aux->getNome ( ) << " matriculado na Disciplina " << nome << "." << endl;

       aux = aux->prox;
   }

}
```

```
void ListaAlunos::listeAlunos2 ( )
{
   ElAluno* paux = NULL;
   paux = pElAlunoAtual;

   while ( paux != NULL )
   {
     // printf (" Aluno %s matriculado na Disciplina %s \n", paux->getNome(), nome);

     cout << " Aluno " << paux->getNome ( ) << " matriculado na Disciplina " << nome << "." << endl;

     paux = paux->pAnte;
   }

}
```

```
#ifndef _DISCIPLINA_H_
#define _DISCIPLINA_H_

#include "ListaAlunos.h"
#include "Departamento.h"

class Disciplina
{
private:
   int id;
   char nome [150] ;
   char area_conhecimento [150];

   Departamento* DeptoAssociado;

   ListaAlunos ObjLAlunos;

 public:

   Disciplina ( int na = 45, char* ac = " " );
   ~Disciplina ( );

   Disciplina* pProx;
   Disciplina* pAnte;

   void setId ( int i );
   int getId ( );

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

   void setDepartamento ( Departamento* pd );
   Departamento* getDepartamento ( );

   void incluaAluno ( Aluno*  pa );
   void listeAlunos ( );
   void listeAlunos2 ( );
};

#endif
```

```
#include "stdafx.h"
#include "Disciplina.h"

Disciplina::Disciplina ( int na, char* ac ):
ObjLAlunos ( na, ac )
{
   pDeptoAssociado          = NULL;
   pProx                    = NULL;
   pAnte                    = NULL;
   strcpy ( area_conhecimento, ac );
}

Disciplina::~Disciplina ( )
{ pDeptoAssociado          = NULL;
  pProx                    = NULL;
  pAnte                    = NULL;
}

. . .

void Disciplina::setDepartamento (Departamento* pd)
{
  pDeptoAssociado = pd;
  pd->setDisciplina ( this );
}

void Disciplina::incluaAluno ( Aluno* pa )
{
   ObjLAlunos.incluaAluno ( pa );
}

void Disciplina::listeAlunos ( )
{
   ObjLAlunos.listeAlunos ( );
}

void Disciplina::listeAlunos2 ( )
{
   ObjLAlunos.listeAlunos2 ( );
}
```

Exercício 1

- Cada Departamento deve ser capaz de armazenar uma lista de disciplinas.

- A classe *Disciplina*, entretanto, não deverá possuir um ponteiro para o Próximo. Isto deverá estar em uma classe associada chamada *ElDisciplina relacionada a ListaDisciplina*...

Obs.: Solução em Diagrama de Classes da UML e em Código C++

```
#ifndef _DEPARTAMENTO_H_
#define _DEPARTAMENTO_H_

class Universidade;
class Disciplina;
class ListaDisciplinas;

class Departamento
{

 private:

    char nome [ 130 ];

    // Associação para com uma Universidade.
    Universidade* pUniv;
    // Associação para com várias Disciplinas.
    ListaDisciplinas* pObjLDisciplinas;

 public:
    Departamento ( );
    ~Departamento ( );

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

    void setUniversidade ( Universidade* pu );
    Universidade* getUniversidade ( );

    void incluaDisciplina ( Disciplina* pdi );
    void listeDisciplinas ( );
    void listeDisciplinas2 ( );

};
#endif
```

```
#ifndef _DISCIPLINA_H_
#define _DISCIPLINA_H_
#include "ListaAlunos.h"
#include "Departamento.h“

class Disciplina
{
private:

   int id;
   char nome [ 150 ];
   char area_conhecimento [ 150 ];

   Departamento* pDeptoAssociado;
   ListaAlunos       ObjLAlunos;

 public:

   Disciplina (  int na = 45, char* ac = "" );
   ~Disciplina ( );

   void setId ( int i );
   int    getId ( );

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

   void setDepartamento ( Departamento* pd );
   Departamento* getDepartamento ( );

   void incluaAluno ( Aluno* pa );
   void listeAlunos ( );
   void listeAlunos2 ( );
};
#endif
```

```
#include "stdafx.h"
#include "Departamento.h"
#include "Universidade.h"
#include "Disciplina.h“
#include "ListaDisciplinas.h"

Departamento::Departamento ( )
{
     pObjLDisciplinas = new ListaDisciplinas ( -1, " " );
}

Departamento::~Departamento ( )
{
     if ( pObjLDisciplinas )
     {
          delete pObjLDisciplinas;
     }
}

void Departamento::setNome ( char* n )
{
   strcpy ( nome, n );
   pObjLDisciplinas->setNome ( n );
}

char* Departamento::getNome ( )
{
   return nome;
}


void Departamento::setUniversidade ( Universidade* pu )
{
  pUniv = pu;
}
```

```
Universidade* Departamento::getUniversidade ( )
{
   return pUniv;
}

void Departamento::incluaDisciplina ( Disciplina* pdi )
{
    pObjLDisciplinas->incluaDisciplina ( pdi );
}

void Departamento::listeDisciplinas ( )
{
    pObjLDisciplinas->listeDisciplinas ( );
}

void Departamento::listeDisciplinas2 ( )
{
     pObjLDisciplinas->listeDisciplinas2 ( );
}
```

```
#ifndef _LISTADISCIPLINAS_H_
#define _LISTADISCIPLINAS_H_

#include "ElDisciplina.h"
#include "Disciplina.h"

class ListaDisciplinas
{
private:

   int cont_disc;
   int numero_disc;
   char nome [ 150 ];

   ElDisciplina *pElDisciplinaPrim;
   ElDisciplina *pElDisciplinaAtual;

public:

   ListaDisciplinas ( int nd = 1000, char* n = "" );
   ~ListaDisciplinas ( );

   void setNome ( char* n );
   void incluaDisciplina ( Disciplina* pdi );

   void listeDisciplinas ( );
   void listeDisciplinas2 ( );
};

#endif
```

```
#include "stdafx.h"
#include "ListaDisciplinas.h"

ListaDisciplinas::ListaDisciplinas ( int nd, char* n )
{
    numero_disc   = nd;
    cont_disc     = 0;

    pElDisciplinaPrim  = NULL;
    pElDisciplinaAtual = NULL;

    strcpy ( nome, n );
}

ListaDisciplinas::~ListaDisciplinas ( )
{
  ElDisciplina *paux1, *paux2;

  paux1 = pElDisciplinaPrim;
  paux2 = paux1;

  while ( paux1 != NULL )
  {
      paux2 = paux1->pProx;
      delete ( paux1 );
      paux1 = paux2;
  }

  pElDisciplinaPrim  = NULL;
  pElDisciplinaAtual = NULL;
}


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

```
void ListaDisciplinas::incluaDisciplina ( Disciplina* pdi )
{

   if (
        ( ( cont_disc < numero_disc ) && ( pdi != NULL) ) ||
        ( ( numero_disc == -1 )       && ( pdi != NULL) )
      )
   {
      // Aqui é criado um ponteiro para LAluno
      ElDisciplina* paux;
      // Aqui é criado um objeto LAluno, sendo seu
      //  endereço  armazenado em aux
      paux = new ElDisciplina ( );

      // Aqui recebe uma cópia do objeto interm.
      paux->setDisciplina ( pdi );

      if ( pElDisciplinaPrim == NULL )
      {
         pElDisciplinaPrim   = aux;
         pElDisciplinaAtual  = aux;
      }
      else
      {
         pElDisciplinaAtual->pProx = paux;
         paux->pAnte               = pElDisciplinaAtual;
         pElDisciplinaAtual        = paux;
      }
      cont_disc++;

   }
   else
   {
       cout << " Disciplina não incluída "
            << " Quantia de disc. já lotada em "
            << numero_disc << " disciplinas." << endl;
   }
}
```

```
void ListaDisciplinas::listeDisciplinas()
{
   ElDisciplina* paux;
   paux = pElDisciplinaPrim;

   while ( paux != NULL)
   {
        cout << "  Disciplina " << paux->getNome ( )
             << "  do deparatamento " << nome << "." << endl;
        paux = paux->pProx;
   }

}

void ListaDisciplinas::listeDisciplinas2 ( )
{
   ElDisciplina* paux;
   paux = pElDisciplinaAtual;

   while ( paux != NULL )
   {
     cout << " Disciplina " << paux->getNome()
          << " do Departamento " << nome << "." << endl;
     paux = paux->pAnte;
   }

}
```

```
#ifndef _ELDISCIPLINA_H_
#define _ELDISCIPLINA_H_

#include "Disciplina.h"

class ElDisciplina
{
 private:
    Disciplina* pDisciplina;

 public:
    ElDisciplina( );
    ~ElDisciplina( );

    ElDisciplina *pProx;
    ElDisciplina *pAnte;

    void setDisciplina (Disciplina* pdi );
    Disciplina* getDisciplina ( );

    char* getNome();
};

#endif
```

```
#include "stdafx.h"
#include "ElDisciplina.h"

ElDisciplina::ElDisciplina ( )
{
    pProx = NULL;
    pAnte = NULL;
}

ElDisciplina::~ElDisciplina ( )
{
    pProx = NULL;
    pAnte = NULL;
}

void ElDisciplina::setDisciplina ( Disciplina* pdi)
{
   pDisciplina = pdi;
}

Disciplina* ElDisciplina::getDisciplina ( )
{
   return pDisciplina;
}

char* ElDisciplina::getNome ( )
{
    return pDisciplina->getNome ( );
}
```

Exercício 2

- Cada Universidade deve ser capaz de armazenar uma lista de Departamentos.

- A classe *Departamento*, entretanto, não deverá possuir um ponteiro para o Próximo. Isto deverá estar em uma classe associada chamada *ElDepartamento* relacionada a *ListaDepartamentos...*

Obs.: Solução em Diagrama de Classes da UML e em Código C++

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "ListaDepartamentos.h"

class Universidade
{
  private:
     char nome  [ 130 ];
     ListaDepartamentos ObjLDepartamentos;

  public:
     Universidade ( );
     ~Universidade ( );

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

     void incluaDepartamento ( Departamento* pd );
     void listeDepartamentos ( );
     void listeDepartamentos2 ( );

};

#endif
```

Continuem a solução.

Exercício 3

Até então, os objetos têm sido criados de forma 'estática', dentro da classe principal. Assim sendo, todos os objetos criados são estabelecidos *à priori*.

Elabore uma solução com um *menu*, no qual o usuário pode escolher que elementos quer 'cadastrar' (que objetos quer criar), por exemplo Universidades, Departamentos, e Disciplinas.

A partir das informações disponibilizadas pelo usuário, instanciar então (dinamicamente) objetos.

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Pessoa.h"
#include "ListaUniversidades.h"
#include "ListaDepartamentos.h"
#include "ListaDisciplinas.h"
#include "Aluno.h"

class Principal
{

 private:
   . . .
   ListaUniversidades      LUniversidades;
   ListaDepartamentos      LDepartamentos;
   ListaDisciplinas        LDisciplinas;

  public:

   Principal ( int dia, int mes, int ano );

   // Inicializações...
   void Inicializa ( );
   . . .
   void Executar ( );
   . . .
   void CadDisciplina ( );
   void CadDepartamento ( );
   void CadUniversidade ( );

    void Menu ( );
    void MenuCad ( );
    void MenuExe ( );
};
#endif
```

```
void Principal::Executar ( )
{
     . . .
     Menu ( );
}
```

```
void Principal::Menu()
{
   int op = -1;

   while (op != 3)
   {
      system ("cls");
      cout << "  Informe sua opção:      "          << endl;
      cout << "  1 - Cadastrar.          "          << endl;
      cout << "  2 - Executar.           "          << endl;
      cout << "  3 – Sair.               "          << endl;
      cin >> op;

      switch ( op )
      {
         case 1: {        MenuCad ( );              }
               break;

         case 2: {        MenuExe ( );              }
               break;

         case 3: {        cout << " FIM " << endl;  }
               break;

         default: {       cout <<  "Opção Inválida."  << endl;
                          getchar();
                  }
      }
   }
}
```

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Pessoa.h"
#include "ListaUniversidades.h"
#include "ListaDepartamentos.h"
#include "ListaDisciplinas.h"
#include "Aluno.h"

class Principal
{

 private:
   . . .
   ListaUniversidades      LUniversidades;
   ListaDepartamentos      LDepartamentos;
   ListaDisciplinas        LDisciplinas;

  public:

   Principal ( int dia, int mes, int ano );

   // Inicializações...
   void Inicializa ( );
   . . .
   void Executar ( );
   . . .
   void CadDisciplina ( );
   void CadDepartamento ( );
   void CadUniversidade ( );

    void Menu ( );
    void MenuCad ( );
    void MenuExe ( );
};
#endif
```

```
void Principal::Executar ( )
{
     . . .
     Menu ( );
}
```

```
void Principal::Menu()
{
   int op = -1;

   while (op != 3)
   {
      system ("cls");
      cout << "  Informe sua opção:      "          << endl;
      cout << "  1 - Cadastrar.          "          << endl;
      cout << "  2 - Executar.           "          << endl;
      cout << "  3 – Sair.               "          << endl;
      cin >> op;

      switch ( op )
      {
         case 1: {      MenuCad ( );              }
              break;

         case 2: {      MenuExe ( );              }
              break;

         case 3: {      cout << " FIM " << endl;  }
              break;

         default: {     cout <<  "Opção Inválida."  << endl;
                        system ("Pause");
                  }
      }
   }
}
```

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Pessoa.h"
#include "ListaUniversidades.h"
#include "ListaDepartamentos.h"
#include "ListaDisciplinas.h"
#include "Aluno.h"

class Principal
{

 private:
   . . .
   ListaUniversidades      LUniversidades;
   ListaDepartamentos      LDepartamentos;
   ListaDisciplinas        LDisciplinas;

  public:

   Principal ( int dia, int mes, int ano );

   // Inicializações...
   void Inicializa ( );
   . . .
   void Executar ( );
   . . .
   void CadDisciplina ( );
   void CadDepartamento ( );
   void CadUniversidade ( );

    void Menu ( );
    void MenuCad ( );
    void MenuExe ( );
};
#endif
```

```
void Principal::Executar ( )
{
    . . .
    Menu ( );
}
```

Obs a parte: o comando *system* nos permite introduzir o princípio de API, por não deixar de ser uma (mesmo que simplificadamente).

API (*Application Programming Interface*) permite ao programa acessar uma outra aplicação via uma 'interface'.

Neste caso, o *system* é a interface que permite acessar comandos do DOS.

```
void Principal::Menu()
{
   int op = -1;

   while (op != 3)
   {
      system ("cls");
      cout << "  Informe sua opção:      "          << endl;
      cout << "  1 - Cadastrar.          "          << endl;
      cout << "  2 - Executar.           "          << endl;
      cout << "  3 – Sair.               "          << endl;
      cin >> op;

      switch ( op )
      {
         case 1: {      MenuCad ( );          }
               break;

         case 2: {      MenuExe ( );          }
               break;

         case 3: {      cout << " FIM " << endl;  }
               break;

         default: {     cout <<  "Opção Inválida."  << endl;
                        system ("Pause");
                  }
      }
   }
}
```

```
void Principal::MenuCad ( )
{
   int op = -1;

   while ( op != 4 )
   {
      system ( "cls" ) ;
      cout << "  Informe sua opção:       “        << endl;
      cout << "  1 - Cadastrar Disciplina."        << endl;
      cout << "  2 - Cadastrar Departamentos."  << endl;
      cout << "  3 - Cadastrar Universidade. "    << endl;
      cout << "  4  - Sair.                    "        << endl;
      cin >> op;

      switch ( op )
      {
         case 1 :  {   CadDisciplina ( );                    }
                     break;

         case 2:    {  CadDepartamento ( );                 }
                      break;

         case 3:   {   CadUniversidade ( );                 }
                     break;

         case 4:   {  cout << " FIM " << endl;              }
                     break;

         default: {
                     cout << "Opção Inválida." << endl;
                     getchar();
                  }
      }
   }
}
```

```
void Principal::MenuExe ( )
{
   int op = -1;

   while ( op != 4 )
   {
      system ( "cls" );
      cout << "  Informe sua opção:       "                 << endl;
      cout << "  1 - Listar Disciplinas.     "             << endl;
      cout << "  2 - Listar Departamentos."                << endl;
      cout << "  3 - Listar Universidade. "                << endl;
      cout << "  4 – Sair.                   "             << endl;
      cin >> op;

      switch (op)
      {
         case 1:  {  LDisciplinas.listeDisciplinas ( );
                     fflush (stdin) ;
                     getchar();
                  }   break;
         case 2:  {   LDepartamentos.listeDepartamentos ( );
                     fflush(stdin);
                     getchar();
                  } break;
         case 3:  { LUniversidades.listeUniversidades ( );
                    fflush(stdin);
                    getchar();
                  } break;
         case 4:  { cout << " FIM " << endl; }
                  break;
         default: { cout << "Opção Inválida." << endl;
                  getchar();  }
      }
   }
}
```

```
void Principal::CadUniversidade ( )
{
   char nomeUniversidade [ 150 ];
   Universidade* puniv = NULL;

   cout << "Qual o nome da universidade." << endl;
   cin  >> nomeUniversidade;

   puniv = new Universidade();
   puniv->setNome ( nomeUniversidade );
   LUniversidades.incluaUniversidade ( puniv );
}
```

```
void Principal::CadDepartamento ( )
{
   char nomeUniversidade [ 150 ];
   char nomeDepartamento [ 150 ];
   Universidade* puniv;
   Departamento* pdepart;

    cout << "Qual o nome da universidade do departamento" << endl;
    cin  >> nomeUniversidade;
    puniv = LUniversidades.localizar ( nomeUniversidade );

    if ( puniv != NULL )
    {
                cout << "Qual o nome do departamento" << endl;
                cin >> nomeDepartamento;
                pdepart = new Departamento();
                pdepart->setNome ( nomeDepartamento );
                pdepart->setUniversidade ( puniv );
                LDepartamentos.incluaDepartamento ( pdepart );
                puniv->incluaDepartamento ( pdepart );
    }
     else
    {
                cout << " Universidade inexistente. " << endl;
    }
}
```

```
#ifndef _LISTAUNIVERSIDADES_H_
#define _LISTAUNIVERSIDADES_H_
#include "ElUniversidade.h"
#include "Universidade.h"

class ListaUniversidades
{
private:
    int  cont_univ;
    int  numero_univ;
    char nome [150];
    ElUniversidade *pElUniversidadePrim;
    ElUniversidade *pElUniversidadeAtual;
public:
    ListaUniversidades ( int nu = 1000, char* n = "" );
    ~ListaUniversidades ( );
     void incluaUniversidade ( Universidade* pu );
     void listeUniversidades ( );
     void listeUniversidades2 ( );
     Universidade* localizar ( char* n );
};
#endif
```

```
Universidade* ListaUniversidades::localizar ( char* n )
{
   ElUniversidade* paux;
   paux = pElUniversidadePrim;
   while ( paux != NULL )
   {
       if ( 0 == strcmp ( n, paux->getNome () ) )
       {
            return paux->getUniversidade ();
       }
      paux = paux->pProx;
   }
   return NULL;
}
```

Exercício

Para cada classe existente no sistema, criar um atributo Id que servirá para identificar de forma única um objeto. Assim sendo, a cada objeto criado/instanciado, atribuir um id único para ele.

Elabore uma solução que permita registrar os "objetos" (i.e. suas informações em arquivos) de forma a recuperá-los em uma segunda execução do sistema.