OO – Engenharia Eletrônica

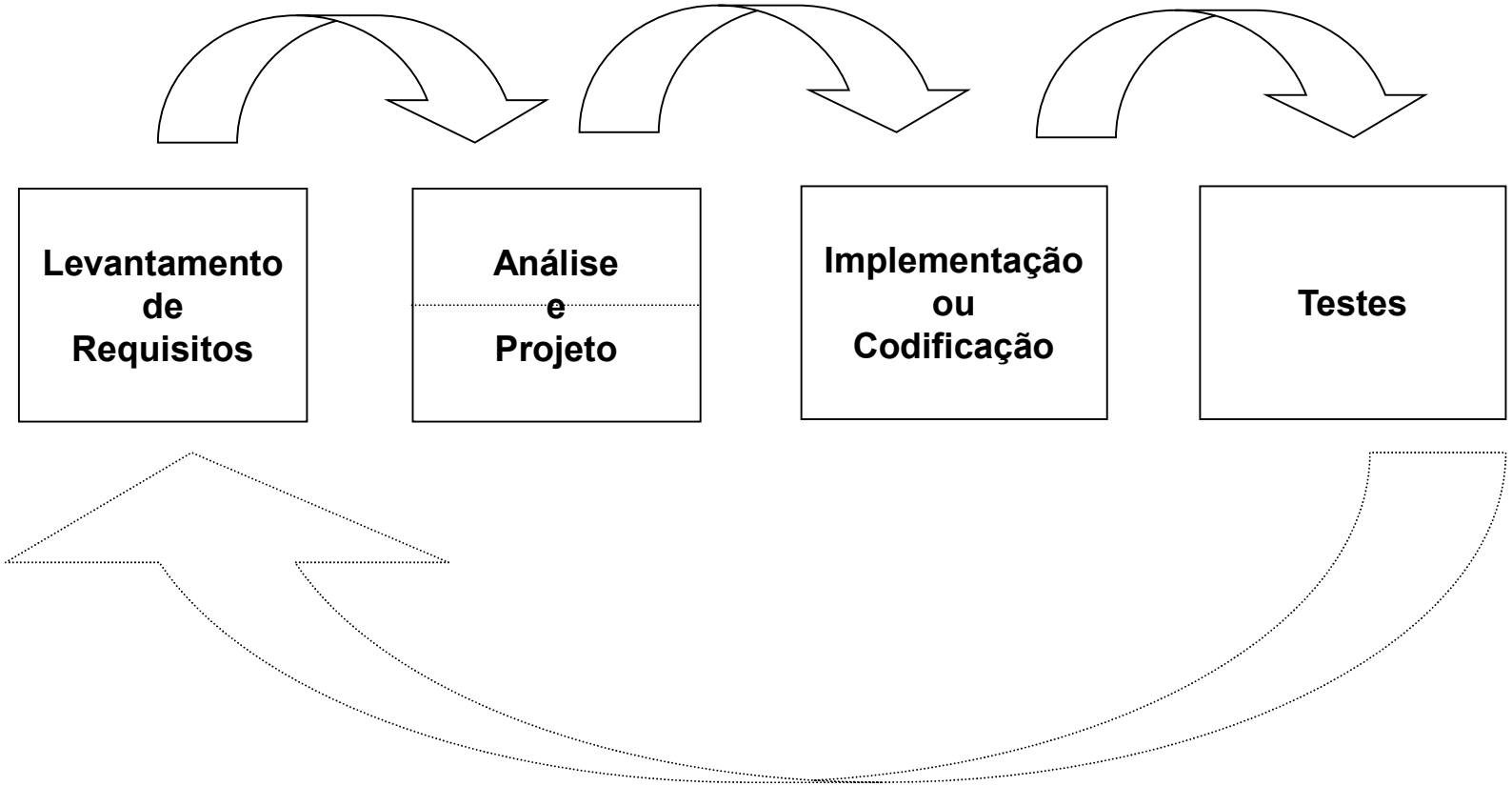
Orientação a Objetos
-
Programação em C++

3º Slides: Relações entre objetos em C++ e *UML*

Prof. Jean Marcelo SIMÃO – DAELN / UTFPR

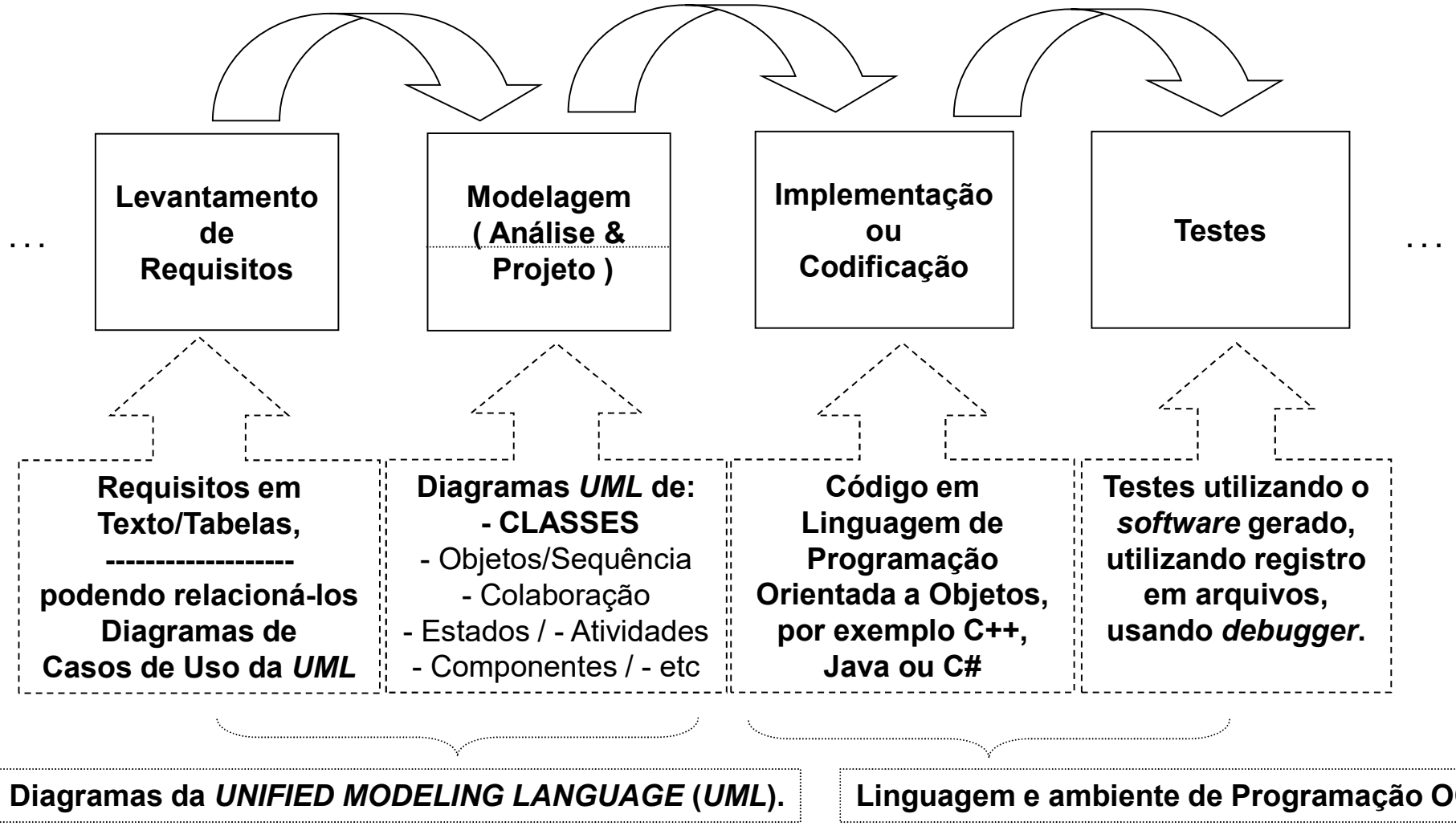
Engenharia de *Software*

Visão Clássica



Engenharia de *Software* OO

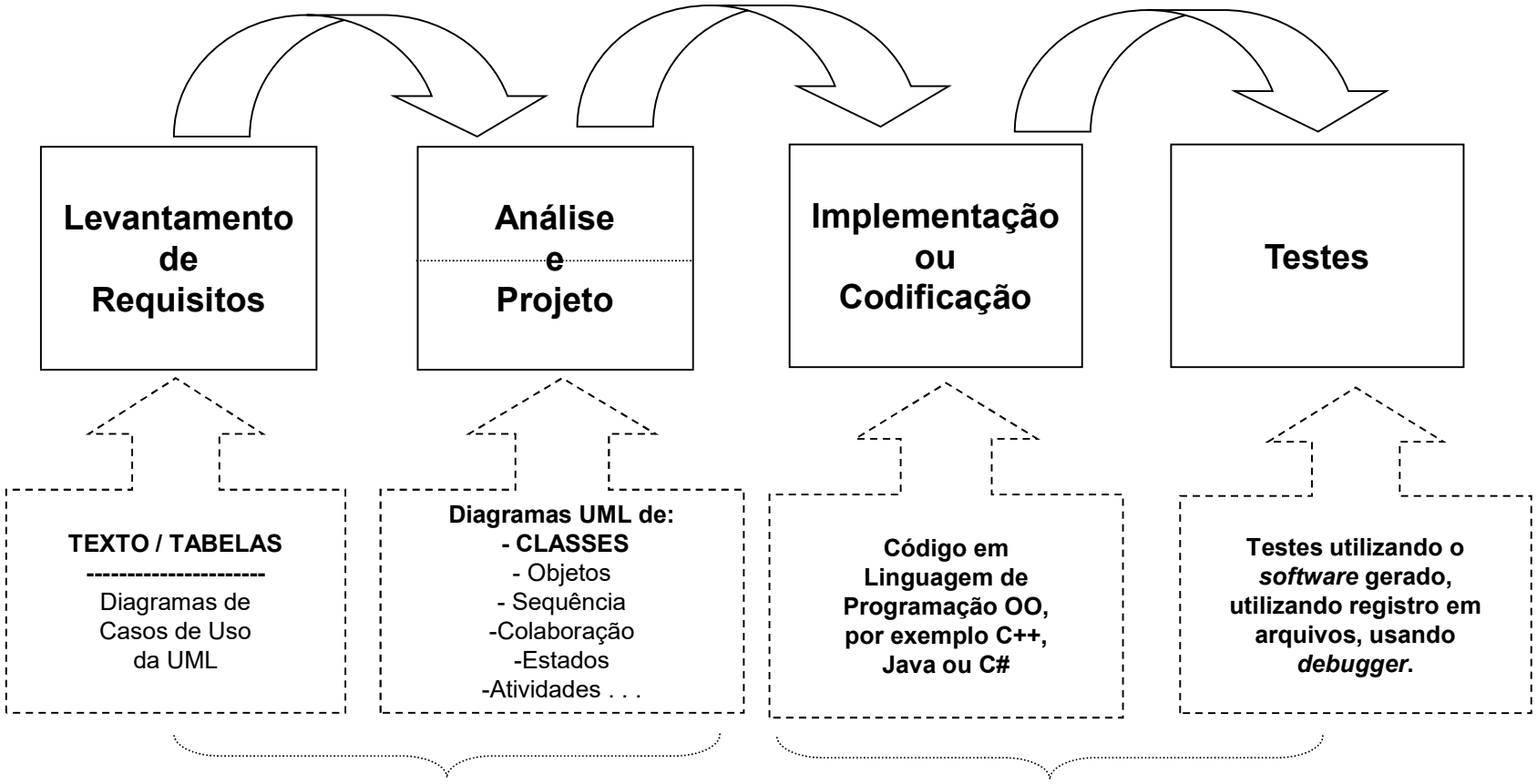
Exemplos de Técnicas



Obs.: Além da UML, há também a SysML (*System Modeling Language*) . . .

Engenharia de *Software* OO

Exemplos de Ferramentas



Ferramentas CASE: StarUML (versão 5.0.2.1517)**, Astah, Jude,** VisualParadigm, Rational Rose**,** System Architect**,** Together**,** Rhapsody **. . .**

Ambiente de Programação OO (integráveis as Ferramentas CASE**): Microsoft Visual Studio (C++), Microsoft Visual C++ .net Express Edition,** Microsoft Visual C++ .net**,** Microsoft Visual C++**,** Borland Builder C++**,** Borland C++**,** **CodeBlocks C++,** Dev C++, G++ **. . .**

Bibliografias

Bibliografias:

- Pressman, R. S. **Software Engineering – A Practitioner's Approach. 6**th Edition McGraw Hill (Higher Education). 2005. ISBN 0-07-285318-2.

- RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual.** 2nd Edition. Addison-Wesley. 2005. ISBN 0-321-26797-4.

- Bezerra, E. **Princípios de Análise e Projeto de Sistemas com UML**. Editora Campus. 2003. ISBN 85-352-1032-6.

- RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Software Development Process.** 1st Edition. Addison-Wesley. 2005. ISBN 0-201-57169-2.

~~**Outras bibliografias:**~~

- ~~GAMMA, E.; HELM, R.; Johnson, R.; Vlissides, J. **Design Patterns: Elements of Reusable Object-oriented Software. Addison Wesle**y Longman, 1995.~~

- ~~Largman, G**. Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design.** Prentice Hall. 1998. ISBN 0-13-748880-7**.**~~

Primeiramente: Qual será ou tem sido nosso exemplo para estudos?

Um sistema acadêmico!

Definindo Requisitos Funcionais (e Gerais) do Sistema Acadêmico

- Registrar no sistema um conjunto de Universidades.
- Registrar no sistema um conjunto de Departamentos relacionados a Universidades.
- Registrar no sistema um conjunto de Disciplinas relacionados a Departamento.
- Registrar um conjunto de Professores relacionados a entidades pertinentes como Departamento e Disciplinas.
- Registrar um conjunto de Alunos relacionados a entidades pertinentes como Departamento e Disciplinas.
- . . .

Obs.: Estes são requisitos funcionais (e gerais) que serão expandidos e refinados (inclusive em termos técnicos) nos enunciados de exercícios. Isto porque estamos no âmbito de uma disciplina aprendendo conceitos progressivamente. Em um sistema real, os requisitos deveriam ser definidos e estabelecidos o quanto antes (dentro do ciclo de engenharia de *software*).

Requisitos Funcionais x Técnicos

- Simplificadamente, requisitos podem ser classificados em Requisitos Funcionais e Requisitos Técnicos.

- Os Requisitos Funcionais são os que definem quais funcionalidade terão o sistemas (sem definir como estas se darão tecnicamente).

- Os Requisitos Técnicos são os que definem como as funcionalidades se darão tecnicamente, salientando meios ou formas.

- Certamente, baseando-se nos requisitos funcionais (sejam gerais ou específicos), pode-se derivar requisitos técnicos.

Retomando a última aula, na qual havia os seguintes ***requisitos técnicos***:

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

- Relacionar a classe Pessoa para com a classe Universidade. Cada objeto de Pessoa poderá ser associado a um objeto de 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 em C++

Universidade.h

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

class Universidade
{
  private:
     char nome[30];

  public:
     // Construtor
     Universidade ( );

     // Destrutor
     ~Universidade ( );

     void setNome ( char* n );

     char* getNome ( );

};
#endif
```

Universidade.cpp

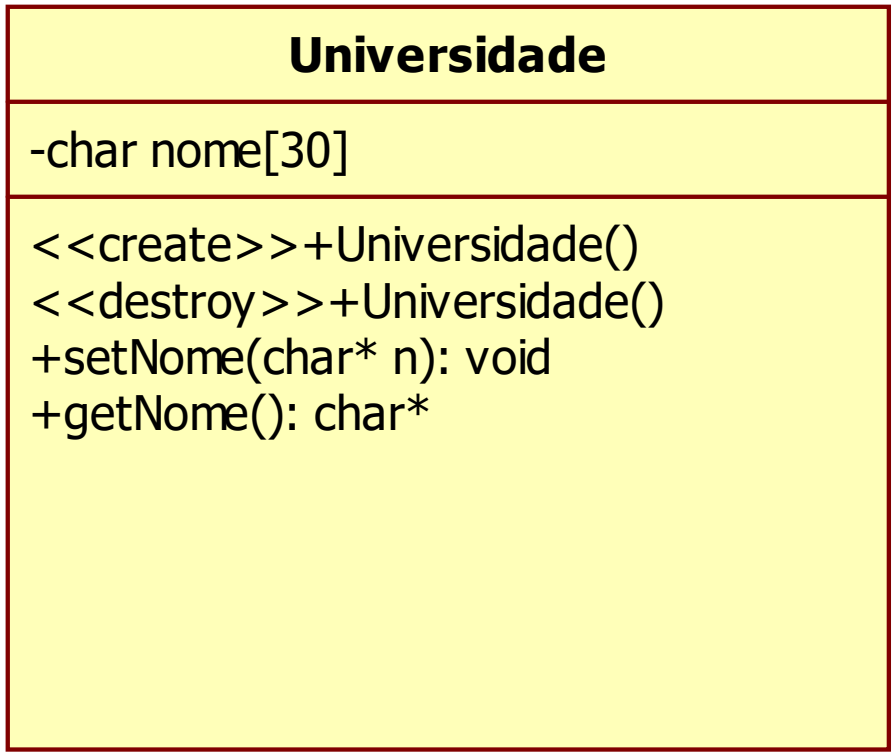
```
#include "Universidade.h"
#include <stdio.h>

Universidade::Universidade ( )
{
   strcpy ( nome, “” );
}

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...

Classe Universidade em UML

Antes de implementar uma classe, pode-se modelar (planejar/projetar) esta classe em UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada).

Isto contribui para respeitar o ciclo de Engenharia de Software: Análise de Requisitos, Análise/Projeto, Implementação e Teste.

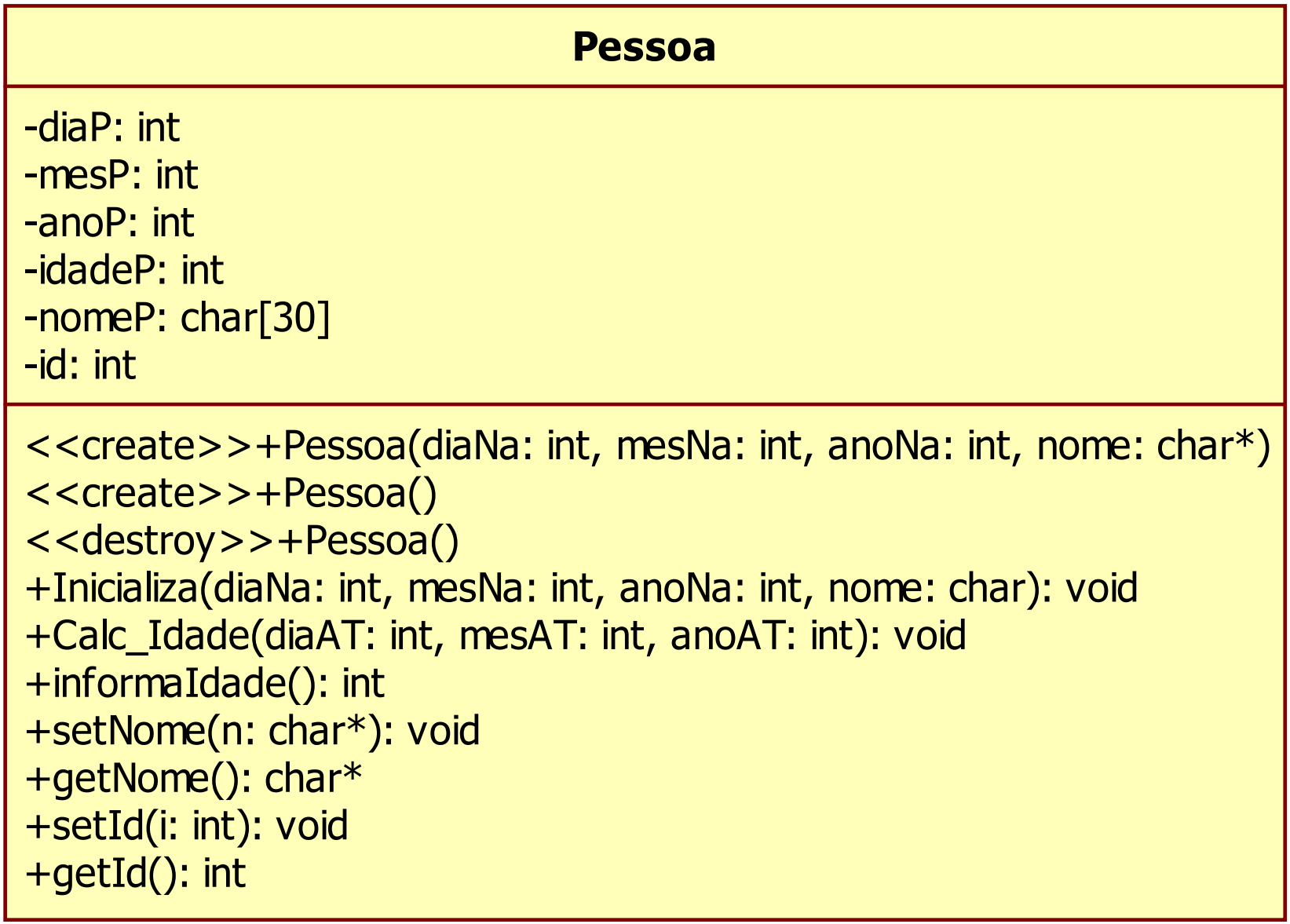


Classe Pessoa em C++ - associada à Universidade

```
#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 ( ) ;
   ~Pessoa ( ) ;

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

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

   // 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 pUnivFiliado é chamado.
    cout <<  nomeP <<" trabalha para a " << pUnivFiliado->getNome() << endl;
}
```

Classe Pessoa em UML



Por exemplo, pode-se utilizar a ferramenta StarUML para planejar/projetar classes

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 em C++

```
#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 );
}
```

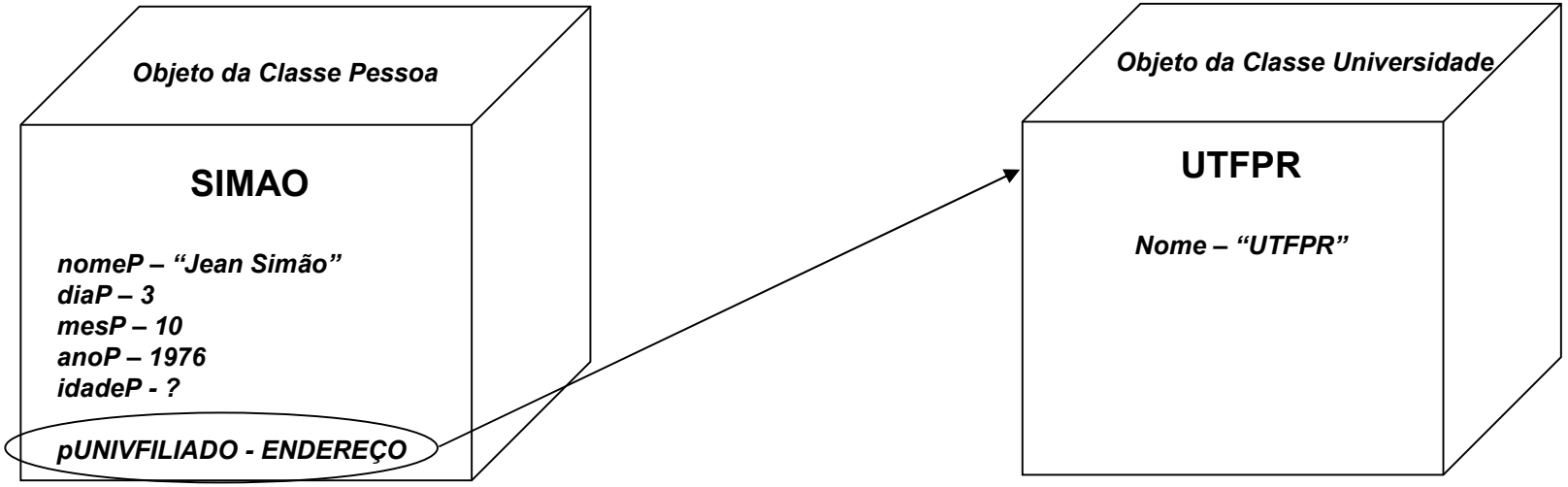
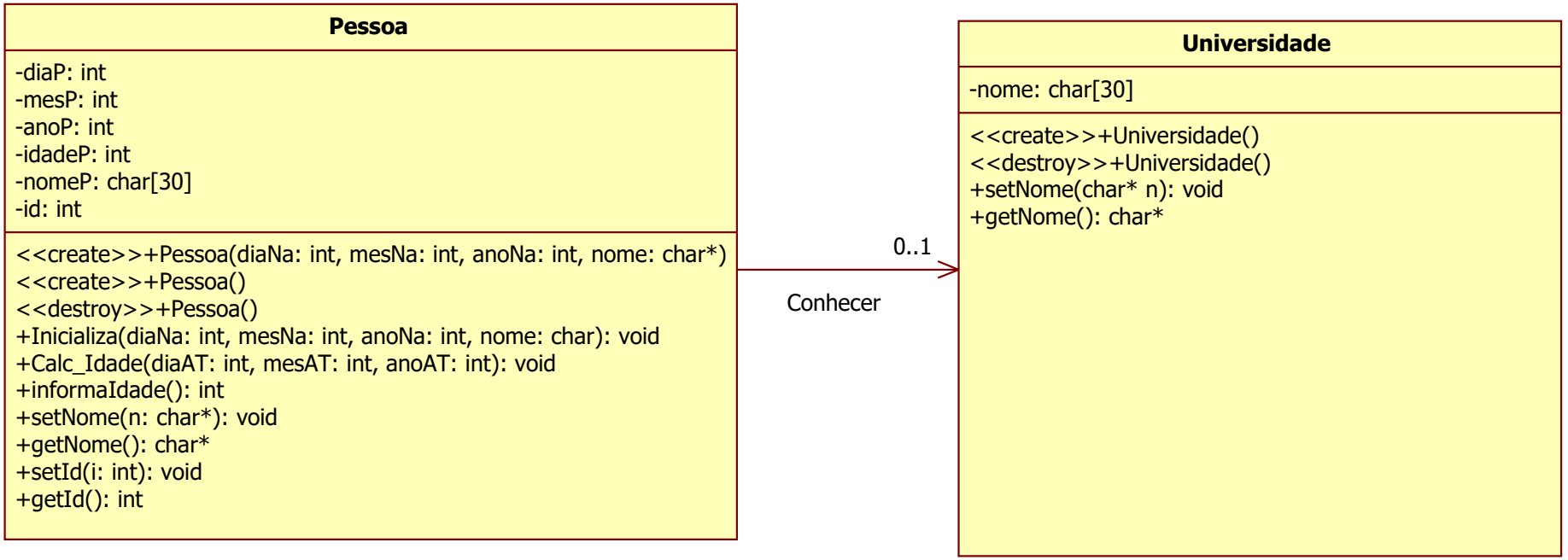


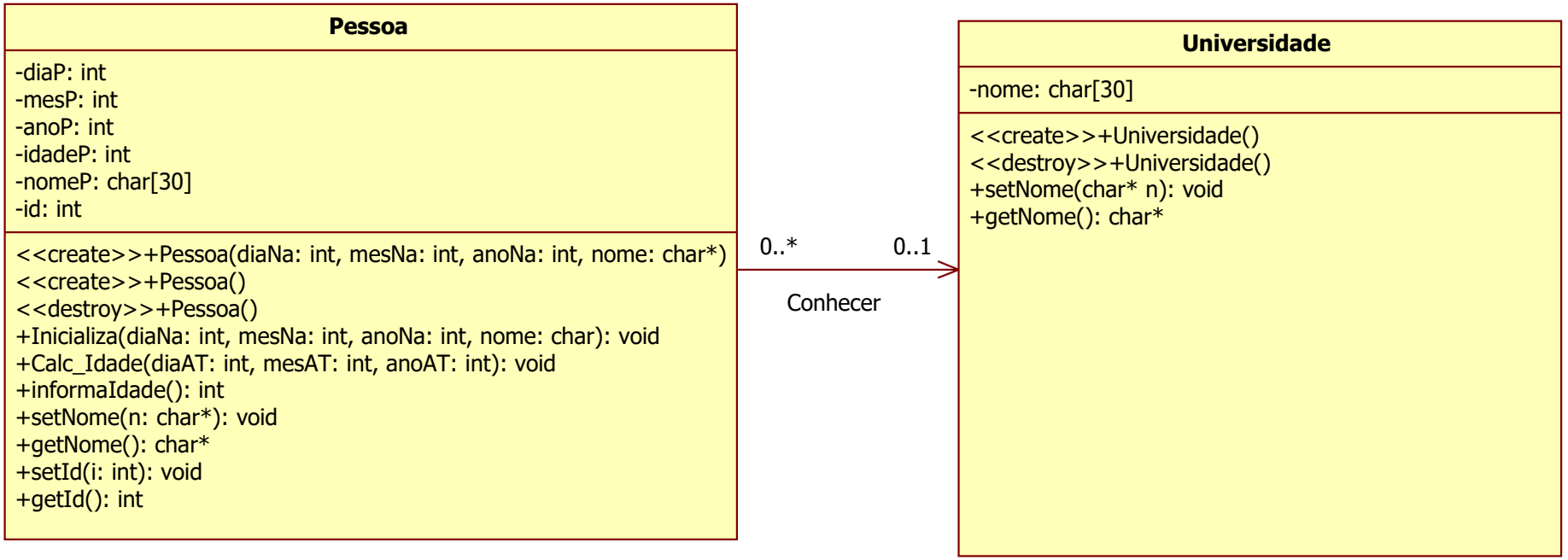
Diagrama de Classes que modela (**analisa**) o relacionamento de associação entre objetos de Pessoa e de Universidade (por meio de suas classes).



Isto é um Diagrama de Classes da UML

Obs.: Tendo em vista que o Diagrama oculta detalhes de implementação, ficando mais próximo do sistema real analisado, ele pode ser considerado como um Diagrama de Classes de Análise.

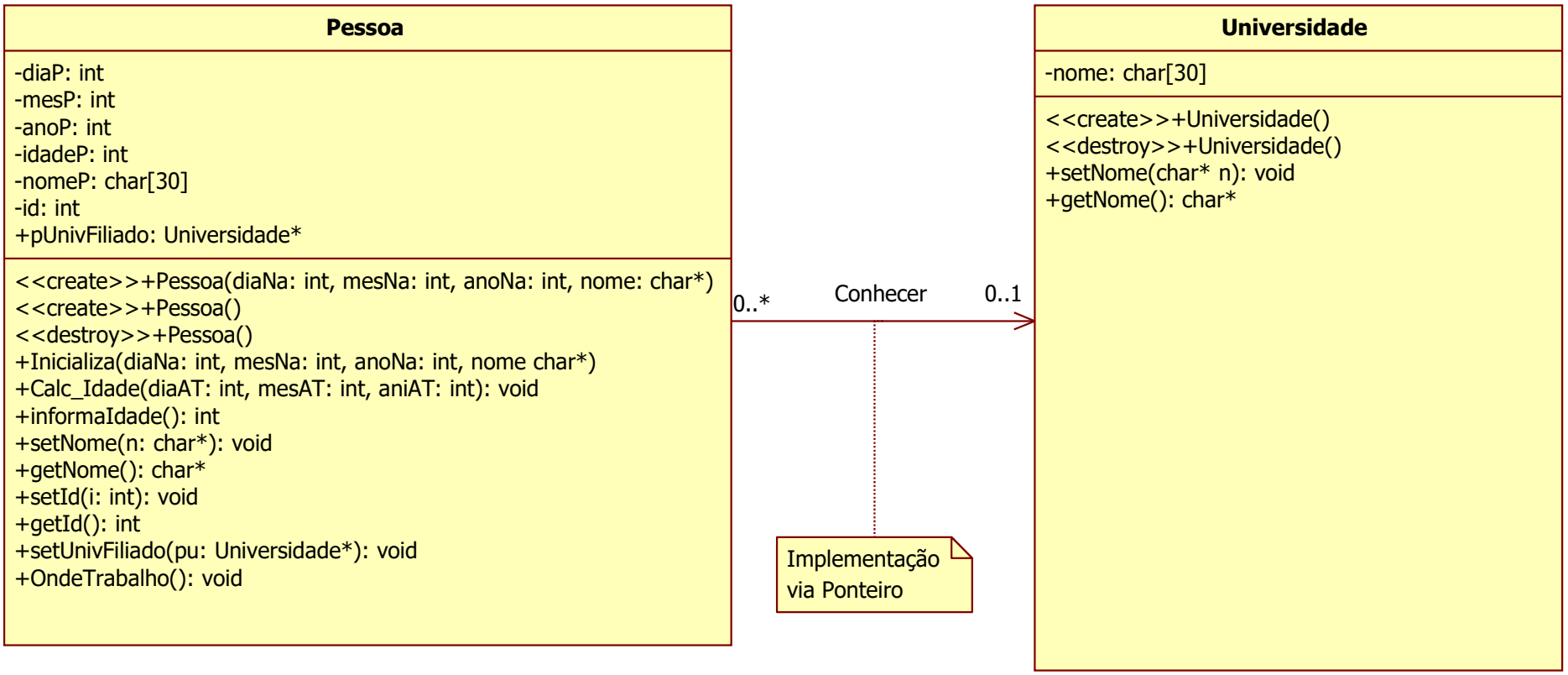
Diagrama de Classes que modela (**analisa**) o relacionamento de associação entre objetos de Pessoa e de Universidade (por meio de suas classes).



Isto é um Diagrama de Classes da UML

Obs.: Tendo em vista que o Diagrama oculta detalhes de implementação, ficando mais próximo do sistema real analisado, ele pode ser considerado como um Diagrama de Classes de Análise.

Diagrama de Classes que modela (**projeta**) o relacionamento de associação entre objetos de Pessoa e de Universidade (por meio de suas classes).



Obs.: Tendo em vista que o Diagrama traz alguns detalhes de implementação, ele pode ser considerado como um <u>Diagrama de Classes de Projeto.</u>

Exercício 1

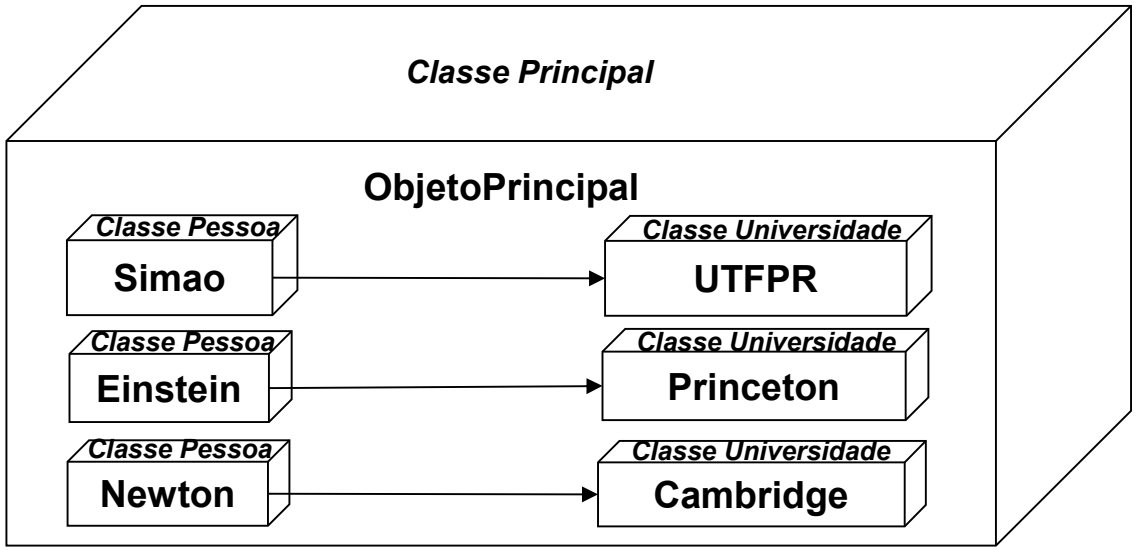
- Criar dois objetos de Universidade associando um para Einstein e outro para Newton.
  - Einstein trabalhou como professor de física em Princeton (Nova Jersey - Estados Unidos da América).
  - Newton trabalhou como professor de matemática em Cambridge (Inglaterra).

Solução - Agregação

```
#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;
    Universidade Princeton;
    Universidade Cambridge;

    int diaAtual;
    int mesAtual;
    int anoAtual;

  public:

    Principal();
    ~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" );
   Princeton.setNome ( "Princeton" );
   Cambridge.setNome( "Cambridge" );

   // Aqui os objetos Universidade e Pessoa são associados.
   Simao.setUnivFiliado ( &UTFPR );
   Einstein.setUnivFiliado ( &Princeton );
   Newton.setUnivFiliado ( &Cambridge );
   . . .
}

Principal::~Principal( )
{
}

void Principal::Executar()
{

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

   Simao.OndeTrabalho ( );
   Einstein.OndeTrabalho ( );
   Newton.OndeTrabalho ( );
}
```

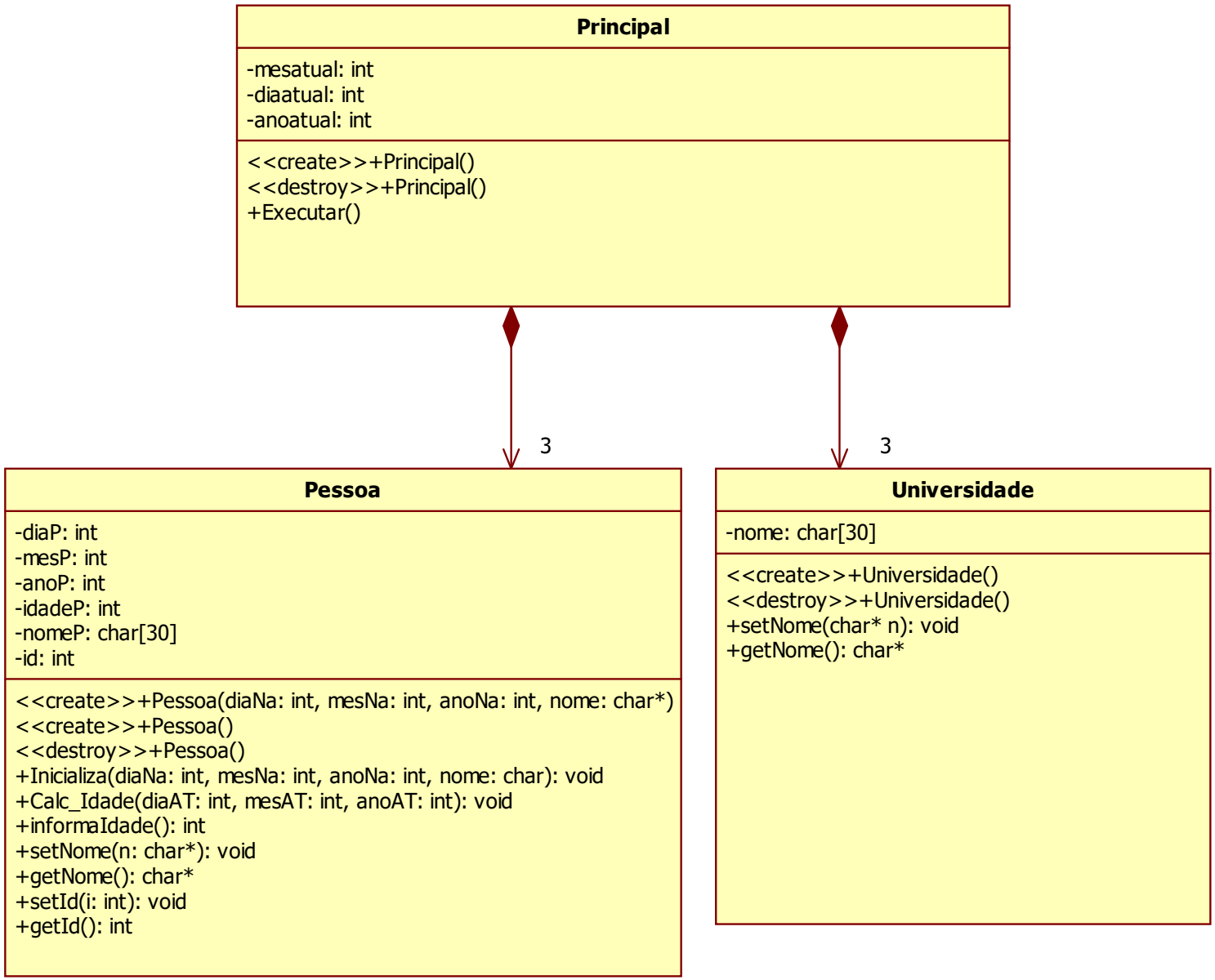
Agregação-forte



O objeto da classe Principal agrega-fortemente (efetivamente) os objeto Simao, Einstein e Newton.

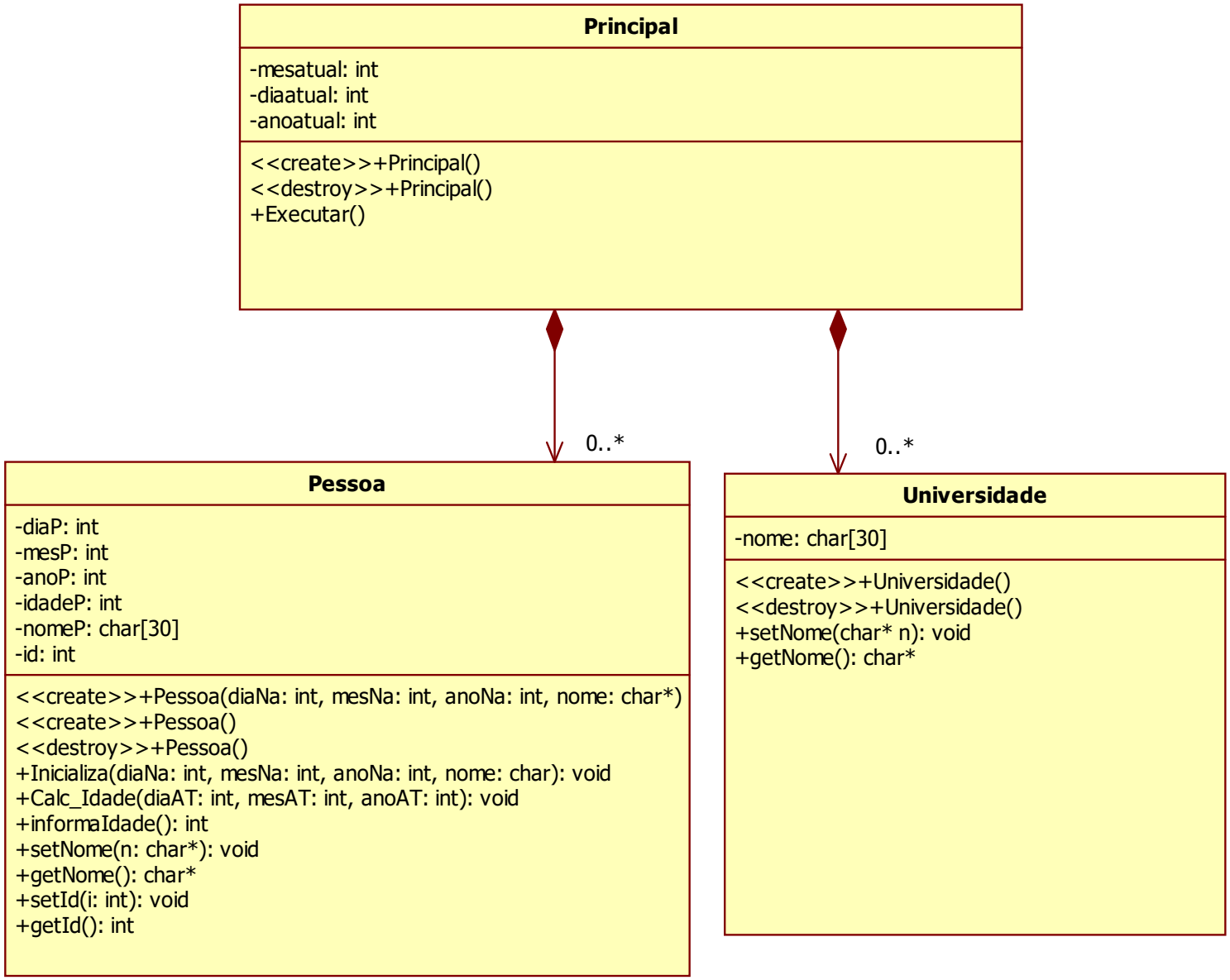
Uma agregação-forte, comumente chamada de COMPOSIÇÃO, é caracterizada pelo fato do objeto agregado só poder ser agregado (efetivamente) por um agregador.

Agregação-forte em UML



A agregação-forte também é chamada de "composição".

Agregação-forte em UML



- O "0..*" é a cardinalidade da agregação-forte (ou composição).
- Um objeto de Principal se compõe de *zero ou mais* objeto de Pessoa.
- Um objeto de Principal se compõe de *zero ou mais* objeto de Universidade

Exercício 2

- Criar uma classe Departamento (em UML e depois em C++) que permita agregar um objeto (de 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.

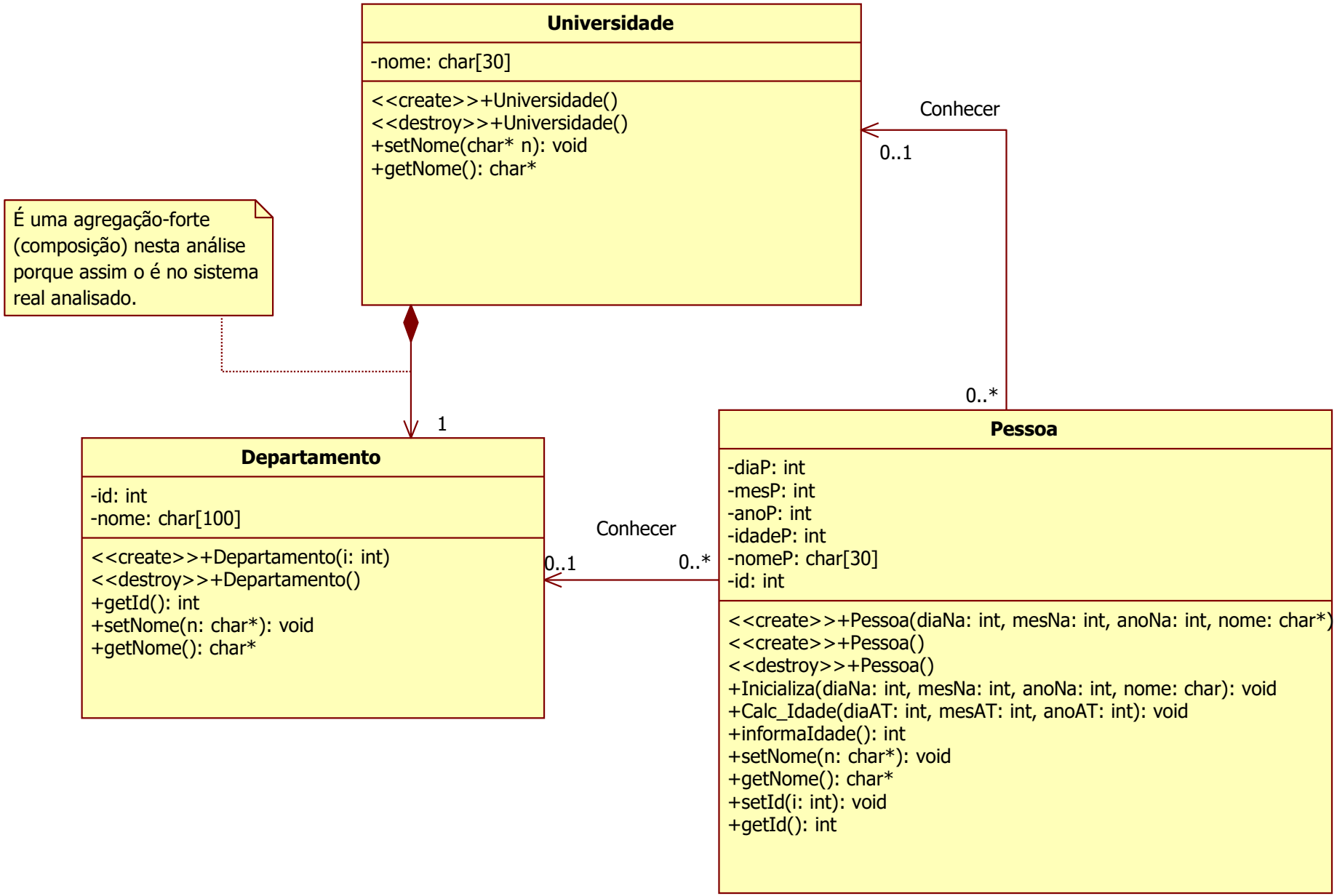
Exercício 2

- Criar uma classe Departamento (em UML e depois em C++) que permita agregar um objeto (de Departamento) na classe Universidade.

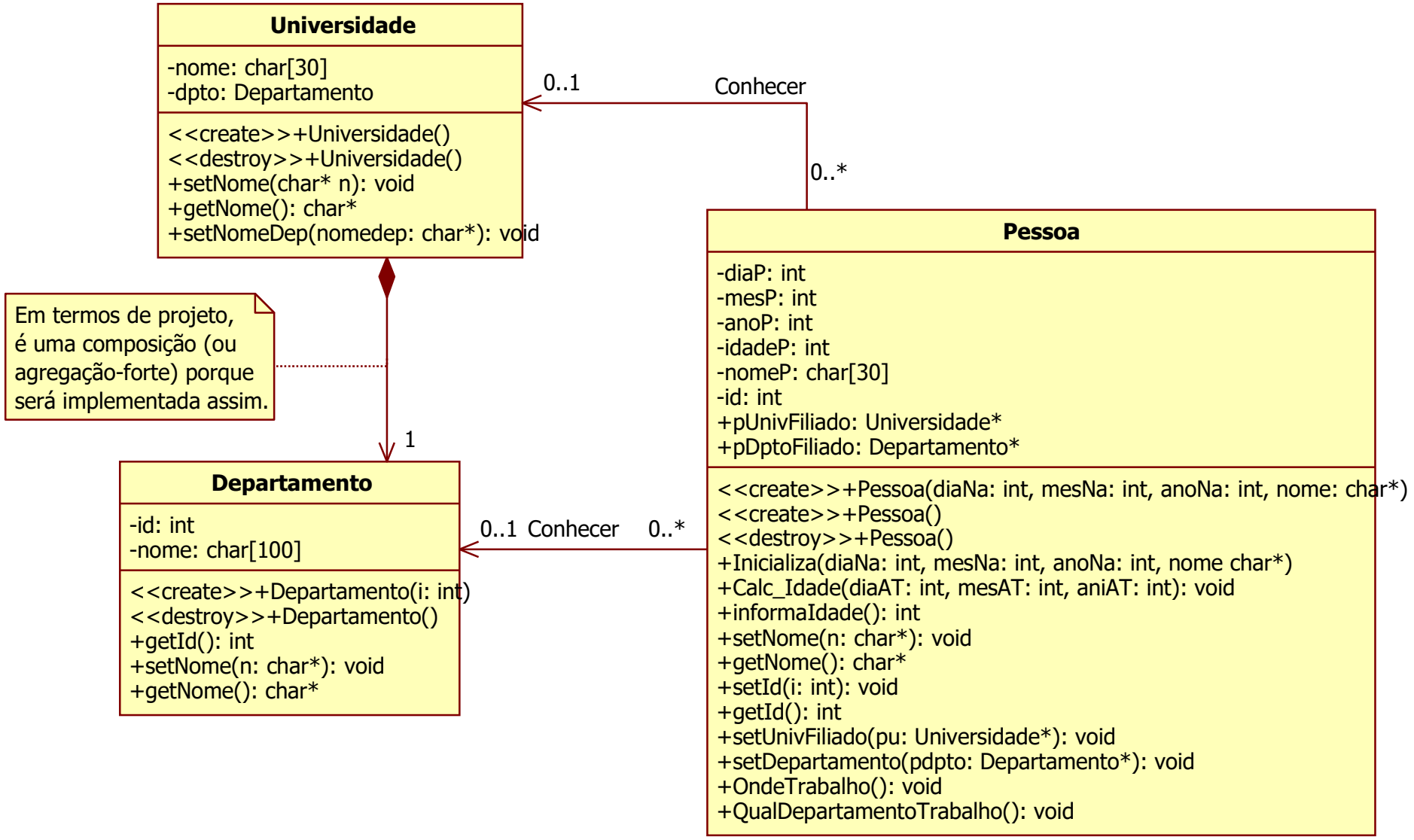
- A classe Pessoa deve possuir uma referência ao departamento que trabalha, ou seja:

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

Modelo em UML - Análise



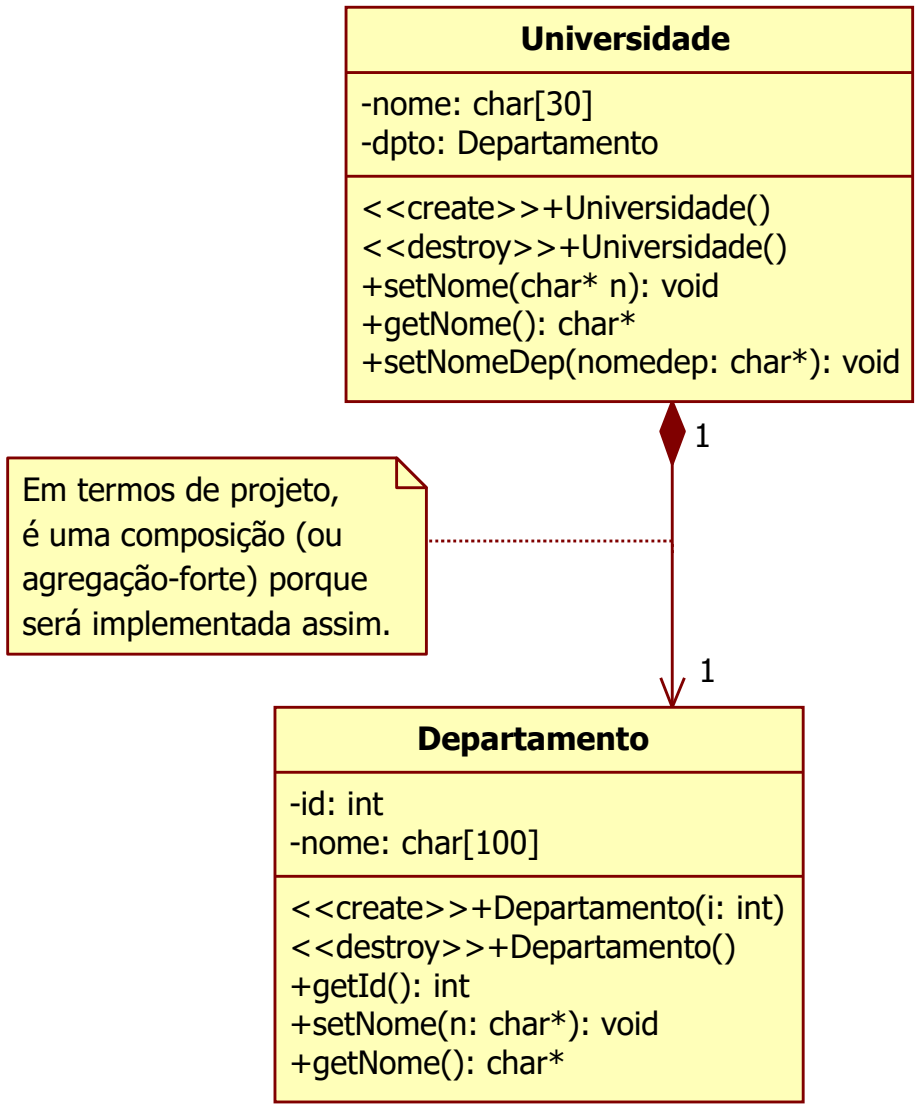
Modelo em UML - Projeto



Modelo em UML – Projeto - Classe Universidade V0

- Solução por agregação-forte ou composição.

- A composição é caracterizada pelo fato do agregado só poder pertencer efetivamente a um agregador.



Obs.: Agregação-forte ou Composição – losango preenchido

Classe Departamento

```
#ifndef _DEPARTAMENTO_H_
#define _DEPARTAMENTO_H_

class Departamento
{
  private:
    char nome[130];

  public:
    Departamento();
    ~Departamento();

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

#endif
```

```
#include "Departamento.h"
// . . .
#include <string.h>

Departamento::Departamento ( )
{
    strcpy ( nome, “” );

}

Departamento::~Departamento ( )
{

}

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

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

Classe Universidade V0

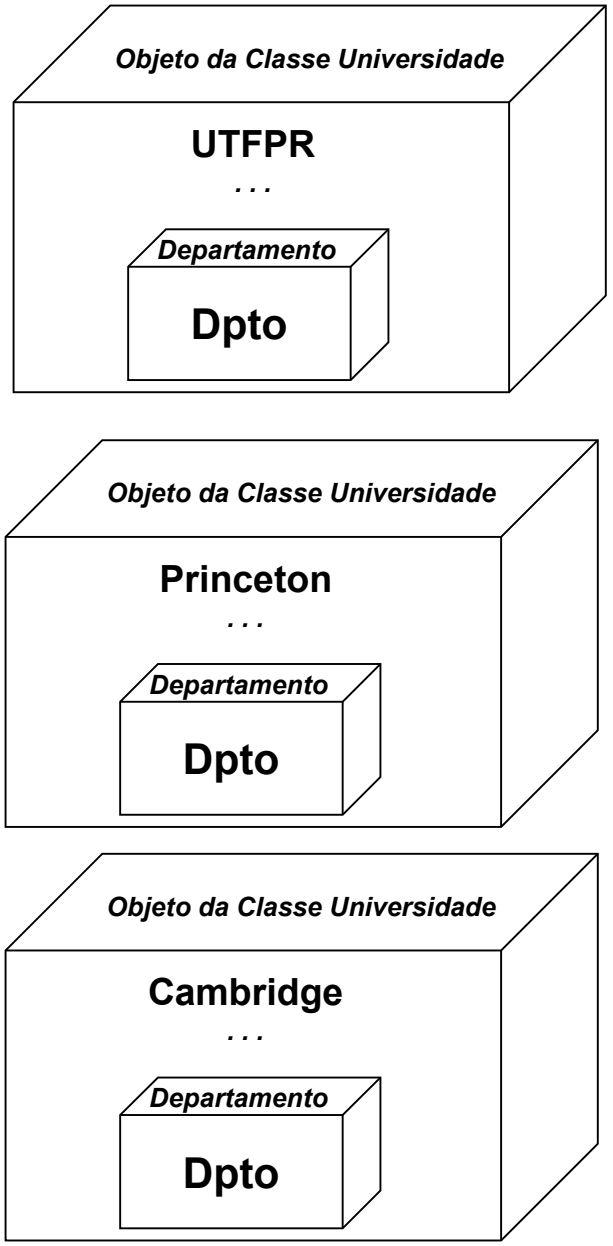
```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome[130];
     Departamento Dpto;    // Objeto Dpto

  public:
     Universidade ();
     ~Universidade ();

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

#endif
```

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

Universidade::Universidade ()
{
      strcpy ( nome, “” );
     //Dpto.setNome("Teste");
}

Universidade::~Universidade ()
{
}

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

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

void Universidade::setNomeDep (char* nome_dep)
{

   Dpto.setnome ( nome_dep );

}
```

Solução boa, mas inflexível.

Classe Universidade V0

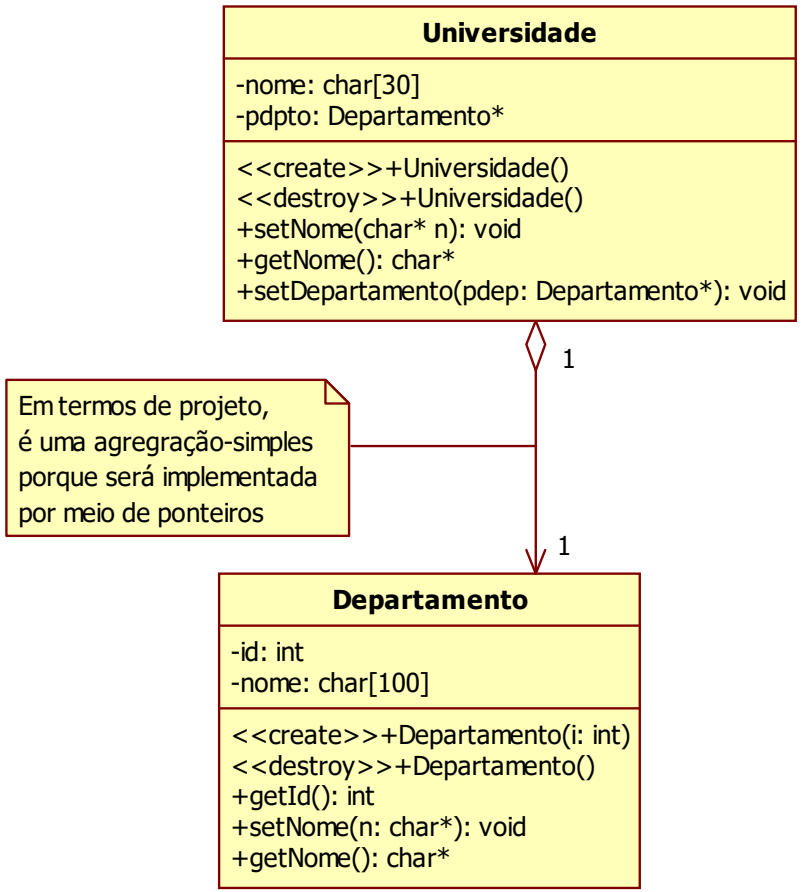
```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

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

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

    Universidade UTFPR;
    Universidade Princeton;
    Universidade Cambridge;


    int diaAtual;
    int mesAtual;
    int anoAtual;

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

#endif
```



Modelo em UML – Projeto - Classe Universidade V1

- Solução por agregação-simples ou fraca.

- A agregação-simples é caracterizada pelo fato do agregado poder <u>eventualmente</u> pertencer (fracamente) a mais de um agregador.



Obs.: Agregação-fraca ou simples – losango vazio

O uso de agregação forte ou fraca causa, por vezes, polêmica.
Mesmo o entender do que é um e outro causa certa polêmica.

Classe Universidade V1

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome[130];
     Departamento* pDpto;        // Ponteiro para departamento

  public:
     Universidade ();
     ~Universidade ();

     void setNome (char* n);
     char* getNome ();
     void setDepartamento (Departamento* pdep);
};

#endif
```

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

Universidade::Universidade ()
{
 pDpto = NULL;
}

Universidade::~Universidade ()
{
}

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

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

void Universidade::setDepartamento (Departamento* pdep)
{

   // Agregação via ponteiros

   pDpto = pdep;

}
```

Solução boa. Ela é “flexível” ...

Classe Universidade V1

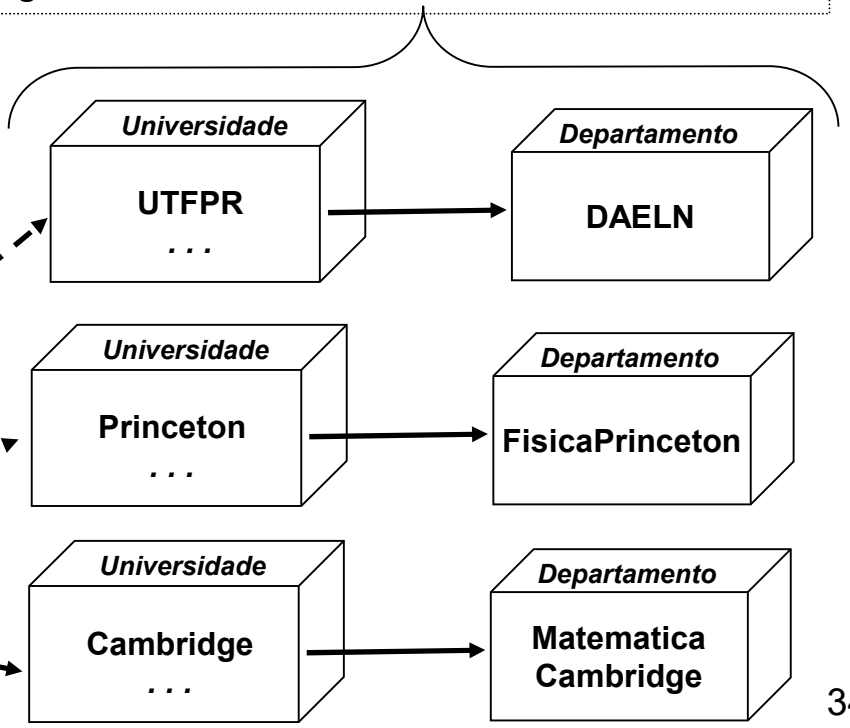
```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_
#include "Pessoa.h"
#include "Universidade.h"
class Principal
{ private:
    Pessoa Simao;
    Pessoa Einstein;
    Pessoa Newton;

    Universidade UTFPR;
    Universidade Princeton;
    Universidade Cambridge;

    Departamento DAELN;
    Departamento FisicaPrinceton;
    Departamento MatematicaCambridge;

    int diaAtual;  int mesAtual; int anoAtual;
  public:
    Principal ();
    void Executar();
};
#endif
```

```
#include "Principal.h"
#include <stdio.h>
Principal::Principal ()
{   . . .
     // Registro do(s) nome(s) do(s) departamento(s)
    DAELN.setNome ("Eletronica");
    FisicaPrinceton.setNome ("Fisica");
    MatematicaCambridge.setNome ("Matematica");

    //"Agregação" do(s) Departamento(s) a(s) Univesidade(s).
    UTFPR.setDepartamento (&DAELN);
    Princeton.setDepartamento (&FisicaPrinceton);
    Cambridge.setDepartamento (&MatematicaCambridge);

}
```

Assim sendo, se um objeto tem ponteiro para outro objeto, isto pode representar:
- uma associação ou
- uma agregação (fraca)
Isto depende do que ele está representando no sistema real.

Entretanto, se um objeto tem explicitamente outro objeto dentro de si (como na V0), então é uma agregação.

Apesar do objeto UTFPR tecnicamente estar associado ao objeto DAELN, teoricamente esta associação representa uma agregação uma vez que no sistema (mundo) real o DAELN está agregado na UTFPR



Exercício 2

- Criar uma classe Departamento (em UML e depois em C++) que permita agregar um objeto (de 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.

Classe Pessoa

```
#ifndef _PESSOA_H_
#define _PESSOA_H_

#include "Universidade.h"

class Pessoa
{
private:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];
   Universidade* pUnivFiliado;
   Departamento* pDptoFiliado;

public:
   Pessoa (int diaNa, int mesNa, int anoNa, char* nome = "");
   Pessoa ( );
   ~Pessoa ( );

   void Inicializa (int diaNa, int mesNa, int anoNa, char* nome = "");
   void Calc_Idade (int diaAT, int mesAT, int anoAT);
   int  informaIdade ();
   void setUnivFiliado (Universidade* pu);
   void setDepartamento (Departamento* pdep);
   void OndeTrabalho ();
   void QualDepartamentoTrabalho ();
};

#endif
```

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

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

Pessoa::Pessoa()
{    }

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

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

int Pessoa::informaIdade ()
{ ... }

void Pessoa::setUnivFiliado (Universidade* pu)
{ ... }

void Pessoa::setDepartamento (Departamento* pdep)
{   // Aqui é uma associação porque é armazenado um apontamento
   // para o Departamento.
   pDptoFiliado = pdep;
}

void Pessoa::OndeTrabalho ()
{
    cout << nomeP << "trabalha para a" << pUnivFiliado->getNome() << endl;
}

void Pessoa::QualDepartamentoTrabalho ()
{
   cout << nomeP << "trabalha para a"
          << pUnivFiliado->getNome()
          << pDptoFiliado->getNome()
          << endl;
}
```

Classe Principal

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

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

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

   Universidade UTFPR;
   Universidade Princeton;
   Universidade Cambridge;

   Departamento DAELN;
   Departamento FisicaPrinceton;
   Departamento MatematicaCambridge;

   int diaAtual;
   int mesAtual;
   int anoAtual;

 public:
   Principal ( );
  ~Principal ( ) { }
   void Executar();
};

#endif
```

```
#include "Principal.h"
#include <stdio.h>
Principal::Principal ()
{   . . .
   // Registro do(s) nome(s) do(s) departamento(s)
   DAELN.setNome ("Eletronica");
   FisicaPrinceton.setNome ("Fisica");
   MatematicaCambridge.setNome ("Matematica");

   // "Agregação" do(s) Departamento(s) a(s) Univesidade(s).
   UTFPR.setDepartamento (&DAELN);
   Princeton.setDepartamento (&FisicaPrinceton);
   Cambridge.setDepartamento (&MatematicaCambridge);

    printf ("\n");
   // "Filiação" a universidade.
   Simao.setUnivFiliado (&UTFPR);
   Einstein.setUnivFiliado (&Princeton);
   Newton.setUnivFiliado (&Cambridge);

   printf("\n");
   // "Filiação" ao departamento.
   Simao.setDepartamento (&DAELN);
   Einstein.setDepartamento (&FisicaPrinceton);
   Newton.setDepartamento (&MatematicaCambridge);
}

void Principal::Executar ()
{ . . .
   printf("\n");
   // Universidade que a Pessoa trabalha.
   Simao.OndeTrabalho();
   Einstein.OndeTrabalho();
   Newton.OndeTrabalho();
   printf("\n");
   // Departamento que a Pessoa trabalha.
   Simao.QualDepartamentoTrabalho ();
   Einstein.QualDepartamentoTrabalho ();
   Newton.QualDepartamentoTrabalho ();
}
```

Atenção

As próximas duas versões (V2 e V3)
são ruins: queira entender o porquê.

Classe Universidade V2

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome[130];
     Departamento Dpto;

  public:
     Universidade ();
     ~Universidade ();

     void setNome (char* n);
     char* getNome ();
     void setDepartamento (Departamento* dep);
};

#endif
```

Solução ruim...

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

Universidade::Universidade ()
{
    //Dpto.setNome("Teste");
}

Universidade::~Universidade ()
{
}

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

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

void Universidade::setDepartamento (Departamento* dep)
{

   Dpto = *dep;

  // Neste caso, o objeto Dpto será um cópia (um clone) do dep passado por
  // parâmetro (por "valor")... Isto pode ser feito quando o objeto apontado por dep
  // não precisa ser sincronizado com o Dpto ou não mudará de valor...

  // Mas muito cuidado com este tipo de construção!

   //printf("Departamento %s está filiado a Universidade %s \n \n", Dpto.getNome(), nome);
}
```

Classe Universidade V3

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome[130];
     Departamento Dpto;

  public:
     Universidade ();
     ~Universidade ();

     void setNome (char* n);
     char* getNome ();
     void setDepartamento (Departamento dep);
};

#endif
```

Solução ruim...

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

Universidade::Universidade ()
{
    //Dpto.setNome("Teste");
}

Universidade::~Universidade ()
{
}

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

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

void Universidade::setDepartamento (Departamento dep)
{

 Dpto = dep;

  // Neste caso, o objeto Dpto será um cópia (um clone) do dep passado por
  // parâmetro (por valor)... Isto pode ser feito quando o objeto dep não precisa
  // ser sincronizado com o Dpto ou não mudará de valor...

  // Mas muito cuidado com este tipo de construção!

   //printf("Departamento %s está filiado a Universidade %s \n \n", Dpto.getNome(), nome);
}
```

Exercício 3

UML / C++

Exercício 3

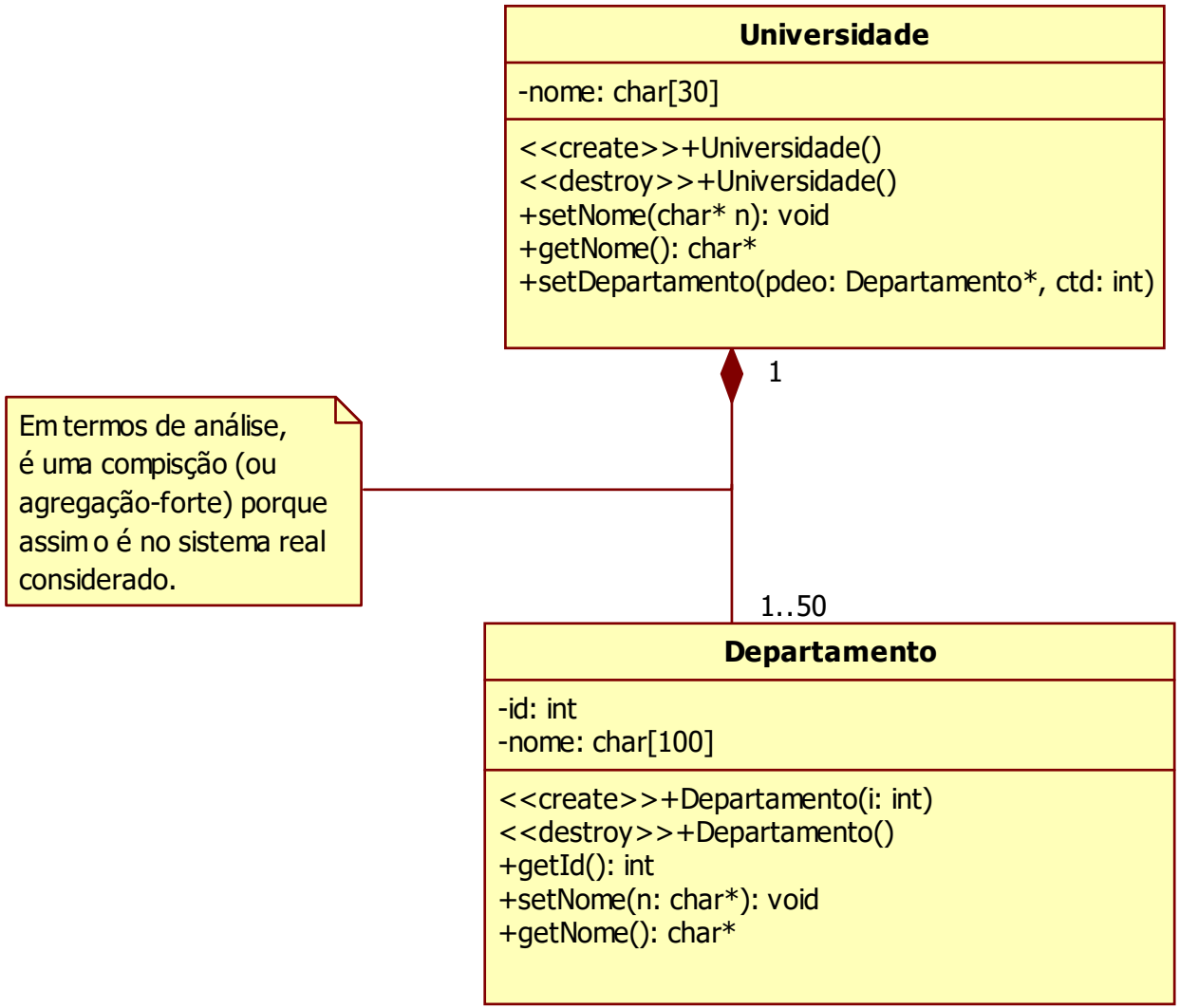
Em diagrama de classe da UML e depois em C++

- Fazer com que uma *Universidade* possa ter até 50 *Departamentos*, tendo pelos menos um.

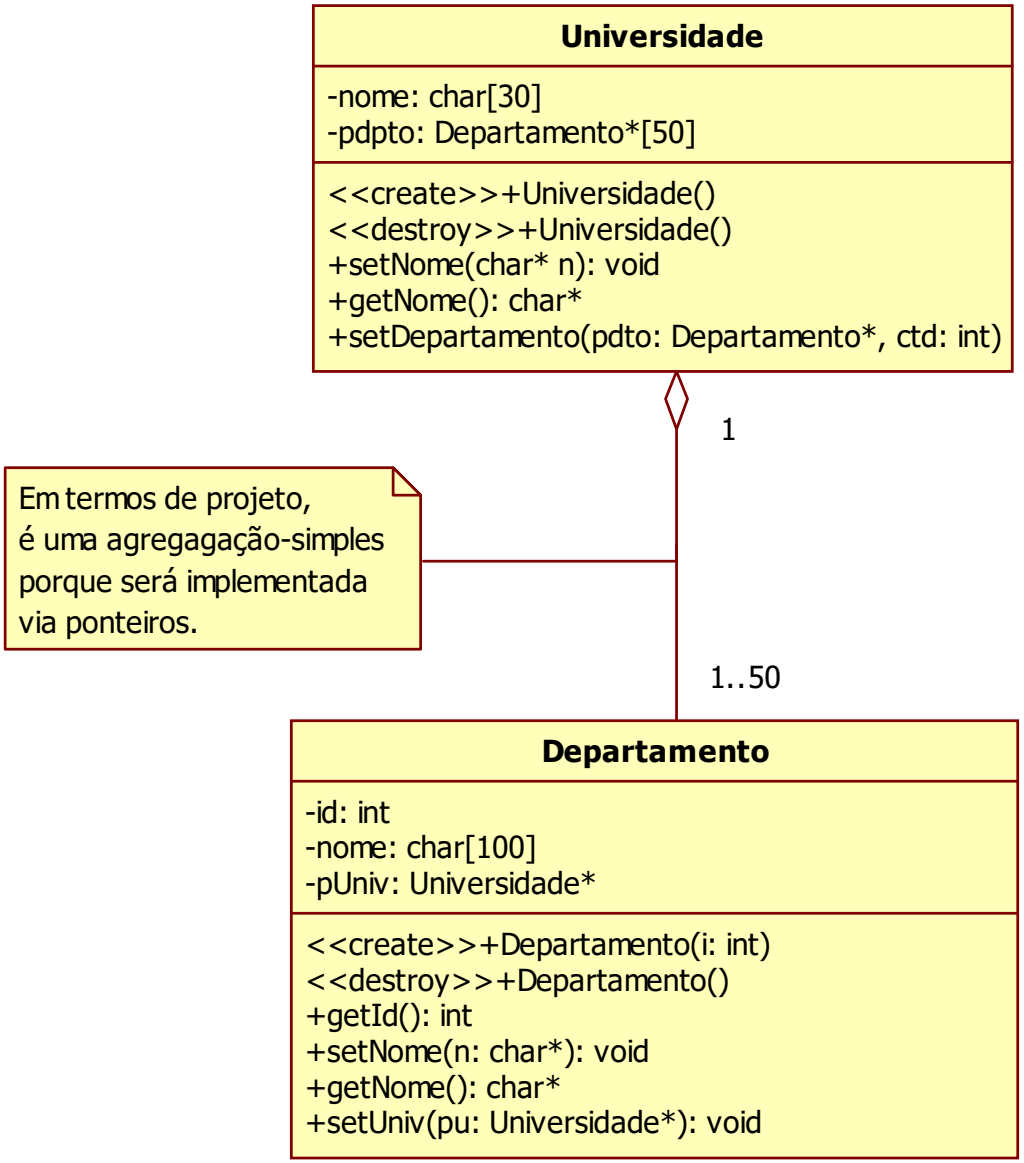
- Fazer com que cada *Departamento* referencie a *Universidade* a qual está filiada.

- Criar mais *Departamentos* filiando-os as *Universidades...*

Modelo em UML - Análise



Modelo em UML - Projeto



Solução para Exercício 1

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome [ 130 ];
      Departamento* pDptos [ 50 ];

public:
     Universidade ( );
     ~Universidade ( );
     void        setNome ( char* n );
     char*       getNome ( );
     void        setDepartamento ( Departamento* pdep, int ctd );
};

#endif
```

Solução para Exercício 1

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
 private:
    char nome [ 130 ];
    Departamento* pDptos [ 50 ];

 public:
    Universidade ( );
    ~Universidade ( );
    void      setNome ( char* n );
    char*     getNome ( );
    void      setDepartamento ( Departamento* pdep, int ctd );
};

#endif
```

Um problema desta implementação é que o número de departamentos é fixo, o que pode gerar desperdício ou sub-dimensionamento.

Há soluções para isto, como uso de lista encadeada ou o uso de alocação dinâmica de vetor com expansão (e.g. desalocação e nova alocação) quando se fizer necessário.

```
#include "stdafx.h"
#include "Universidade.h"
Universidade::Universidade ( )
{
    for ( int i = 0 ; i < 50; i++)   {  pDptos [ i ] = NULL;      }
}

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

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

void Universidade::setDepartamento (  Departamento* pdep , int ctd )
{
  pDptos [ ctd ] = pdep;
}
```

Solução para Exercício 1

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  private:
     char nome [ 130 ];
     Departamento* pDptos [ 50 ];

  public:
     Universidade ( );
     ~Universidade ( );
     void       setNome ( char* n );
     char*      getNome ( );
     void       setDepartamento ( Departamento* pdep, int ctd );
     void       imprimeDptos ( ) ;
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
Universidade::Universidade ( ) {     for ( int i = 0 ; i < 50; i++)   {  pDptos [ i ] = NULL;     }   }

Universidade::~Universidade ( ) { }

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

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

void Universidade::setDepartamento (  Departamento* pdep , int ctd ){
  pDptos [ ctd ] = pdep;
}

void Universidade::imprimeDptos ( ) {
    Departamento*  pDep = NULL;
    for ( int i = 0 ; i < 50; i++)   {
          pDep = pDptos [ i ];
          if ( pDep != NULL )
              cout << pDep->getNome() << endl;
   }
}
```

```
#include "stdafx.h"
#include "Universidade.h"
Universidade::Universidade ( ) {     for ( int i = 0 ; i < 50; i++)   {  pDptos [ i ] = NULL;     }   }

Universidade::~Universidade ( ) { }

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

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

void Universidade::setDepartamento (  Departamento* pdep , int ctd ){
  pDptos [ ctd ] = pdep;
}

void Universidade::imprimeDptos ( ) {  // versão 2
   // ~~Departamento*  pDep = NULL;~~
   for ( int i = 0 ; i < 50; i++)   {
~~        pDep = pDptos [ i ];~~
        if   ( pDptos [ i ] != NULL )   // ~~( pDep != NULL )~~
            cout << pDptos[ i ]->getNome() << endl;
  }
}
```

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_
#include "Pessoa.h"
#include "Universidade.h"

class Principal
{
 private:
 . . .
   Pessoa Christiano;
   Pessoa Diego, Simao;
   Pessoa Einstein;
   Pessoa Newton;

   Universidade UTFPR, Princeton, Cambridge;
   Departamento ModaUFTPR, TecnologiaUTFPR, DAELN, FisicaPrinceton, MatematicaCambridge;
   int diaAtual, mesAtual, anoAtual;

 public:
   Principal ( );
   void Executar ( );
  . . .
};
#endif
```

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

Principal::Principal ( )
{
   . . .   // leitura da data atual

   // Inicialização do(s) ojeto(s) da classe Pessoa
   Christiano.Inicializa          ( 17, 8, 1989, "Zé Maria");
   Diego.Inicializa               (  6, 10, 1989, "Diego");
   Simao.Inicializa               (  3, 10, 1976, "Jean Simão");
   Einstein.Inicializa            ( 14, 3, 1879, "Albert Einstein");
   Newton.Inicializa              ( 4,  1, 1643, "Isaac Newton");

   // Registro do(s) nome(s) da(s) universidade(s)
   UTFPR.setNome ( "Universidade Tecnologica Federal do Paraná");
   Princeton.setNome ( "University of Princeton" );
   Cambridge.setNome  ( "University of Cambridge" );

   // Registro do(s) nome(s) do(s) departamento(s)
   ModaUFTPR.setNome ( "Moda" );
   TecnologiaUTFPR.setNome ( "Tecnologia da UTFPR" );
   DAELN.setNome ( "Eletronica da UTFPR" );
   FisicaPrinceton.setNome ( "Fisica de Princenton" );
   MatematicaCambridge.setNome ( "Matematica de Cambridge" );

   // "Agregação" do(s) Departamento(s) a(s) Univesidade(s).

   UTFPR.setDepartamento    ( &DAELN, 0 );
   UTFPR.setDepartamento    ( &ModaUTFPR, 1 );
   UTFPR.setDepartamento    ( &TecnologiaUTFPR, 2 );
   Princeton.setDepartamento ( &FisicaPrinceton, 0 );
   Cambridge.setDepartamento ( &MatematicaCambridge, 0 );

    // "Filiação" a universidade.
   Christiano.setUnivFiliado ( &UTFPR );
   Diego.setUnivFiliado ( &UTFPR );
   Simao.setUnivFiliado ( &UTFPR );
   Einstein.setUnivFiliado ( &Princeton );
   Newton.setUnivFiliado ( &Cambridge );
   printf ("\n");

// "Filiação" ao departamento.
   Christiano.setDepartamento ( &ModaUFTPR  );
   Diego.setDepartamento ( &TecnologiaUTFPR );
   Simao.setDepartamento ( &DAELN );
   Einstein.setDepartamento ( &FisicaPrinceton );
   Newton.setDepartamento ( &MatematicaCambridge );
   printf ("\n")
}

void Principal::Executar ( )
{
 // Cálculo da idade.
   Christiano.Calc_Idade ( diaAtual, mesAtual, anoAtual );
   Diego.Calc_Idade ( diaAtual, mesAtual, anoAtual );
   Simao.Calc_Idade ( diaAtual, mesAtual, anoAtual );
   Einstein.Calc_Idade ( diaAtual, mesAtual, anoAtual );
   Newton.Calc_Idade ( diaAtual, mesAtual, anoAtual );
   printf ("\n");

// Universidade que a Pessoa trabalha.
   Christiano.OndeTrabalho ( );
   Diego.OndeTrabalho ( );
   Simao.OndeTrabalho ( );
   Einstein.OndeTrabalho ( );
   Newton.OndeTrabalho ( );  printf("\n");

   // Departamento que a Pessoa trabalha.
   Christiano.QualDepartamentoTrabalho ( );
   Diego.QualDepartamentoTrabalho ( );
   Simao.QualDepartamentoTrabalho ( );
   Einstein.QualDepartamentoTrabalho ( );
   Newton.QualDepartamentoTrabalho ( );
}
```

Falta Ainda

- Fazer com que uma *Universidade* possa ter até 50 *Departamentos*.

- **Fazer com que um *Departamento* referencie a *Universidade* a qual está filiado**

- Criar mais *Departamentos* filiando-os as *Universidades..*

Atenção: evite *includes* recursivos!

Universidade.h

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade
{
  . . .
};
#endif
```

Departamento.h

```
#ifndef _DEPARTAMENTO_H_
#define _DEPARTAMENTO_H_

class Universidade;

class Departamento
{
  …
};
#endif
```

Universidade.cpp

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

Universidade::Universidade ( )
{
}

Universidade::~Universidade ( )
{

}
…
```

Departamento.cpp

```
#include <string.h>
#include "Departamento.h“

#include "Universidade.h"

Departamento::Departamento ( )
{
}

Departamento::~Departamento ( )
{
}
…
```

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

Principal::Principal ( )
{
   . . .   // leitura da data atual

   // Inicialização do(s) ojeto(s) da classe Pessoa

   Christiano.Inicializa        ( 17, 8, 1989, "Zé Maria");
   Diego.Inicializa             (  6, 10, 1989, "Diego");
   Simao.Inicializa             (  3, 10, 1976, "Jean Simão");
   Einstein.Inicializa          ( 14, 3, 1879, "Albert Einstein");
   Newton.Inicializa            ( 4,  1, 1643, "Isaac Newton");

   // Registro do(s) nome(s) da(s) universidade(s)

   UTFPR.setNome ( "Universidade Tecnologica Federal do Paraná");
   Princeton.setNome ( "University of Princeton" );
   Cambridge.setNome  ( "University of Cambridge" );

   // Registro do(s) nome(s) do(s) departamento(s)
   ModaUFTPR.setNome ( "Moda" );
   TecnologiaUTFPR.setNome ( "Tecnologia da UTFPR" );
   DAELN.setNome ( "Eletronica da UTFPR" );
   FisicaPrinceton.setNome ( "Fisica de Princenton" );
   MatematicaCambridge.setNome ( "Matematica de Cambridge" );

   // "Agregação" do(s) Departamento(s) a(s) Univesidade(s).
   UTFPR.setDepartamento    ( &DAELN, 0 );
   UTFPR.setDepartamento    ( &ModaUTFPR, 1 );
   UTFPR.setDepartamento    ( &TecnologiaUTFPR, 2 );
   Princeton.setDepartamento ( &FisicaPrinceton, 0 );
   Cambridge.setDepartamento ( &MatematicaCambridge, 0 );


    DAELN.setUniversidade   ( &UTFPR );

    ....

   // "Filiação" a universidade.
    Christiano.setUnivFiliado ( &UTFPR );
    Diego.setUnivFiliado ( &UTFPR );
    Simao.setUnivFiliado ( &UTFPR );
    Einstein.setUnivFiliado ( &Princeton );
    Newton.setUnivFiliado ( &Cambridge );
    printf ("\n");
    // "Filiação" ao departamento.
    Christiano.setDepartamento ( &ModaUFTPR  );
    Diego.setDepartamento ( &TecnologiaUTFPR );
    Simao.setDepartamento ( &DAELN );
    Einstein.setDepartamento ( &FisicaPrinceton );
    Newton.setDepartamento ( &MatematicaCambridge );
    printf ("\n");
}

void Principal::Executar ( )
{  // Cálculo da idade.
    Christiano.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    Diego.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    Simao.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    Einstein.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    Newton.Calc_Idade ( diaAtual, mesAtual, anoAtual );
    printf ("\n");

// Universidade que a Pessoa trabalha.
    Christiano.OndeTrabalho ( );
    Diego.OndeTrabalho ( );
    Simao.OndeTrabalho ( );
    Einstein.OndeTrabalho ( );
    Newton.OndeTrabalho ( );  printf("\n");

    // Departamento que a Pessoa trabalha.
    Christiano.QualDepartamentoTrabalho ( );
    Diego.QualDepartamentoTrabalho ( );
    Simao.QualDepartamentoTrabalho ( );
    Einstein.QualDepartamentoTrabalho ( );
    Newton.QualDepartamentoTrabalho ( );
}
```