

Uma Ferramenta para Automação do Processo de Projeto e Implementação de Classes em Ambientes Orientados por Objetos

Mark Alan J. Song
Roberto S. Bigonha
Mariza A. S. Bigonha

CLEI99 - Assunção - Paraguai

Metodologia

- Objetos
 - Características - estado do objeto
 - Comportamentos - operações
- Classes
 - Objeto - instância de uma classe
- Hierarquias de Classes
 - Todo-Parte (cliente/servidor)
 - Generalização-Especialização (ascendente/descendente)

Projeto Orientado por Objetos

- Construção de sistemas como uma *coleção estruturada* de **tipos abstratos de dados**.
- Cada módulo implementa uma abstração
- Coleção - reúso
- Estruturada - relação

Etapas

- Identificação dos Objetos
 - “substantivos”
 - atributos/operações
- Classes
 - dificuldades:
 - atributos/comportamentos - abstração não identificada: criar novas classes
 - atributo descrito como membro de dado ou calculado: novo comportamento

Etapas...

- Definição de Classes Desnecessárias
 - a posição de um avião
- Critérios
 - atributos/funcionalidades comuns
 - completeza: operações eficientes e significativas
 - genericidade: uso por qualquer cliente
 - primitividade

Etapas...

- Identificar as Dependências
 - relações, em geral, resultam nas dependências das interfaces
 - *Círculos/Ponto*
 - nem sempre são visíveis na interface
- Definir as Hierarquias
 - organização em hierarquias
 - dificuldades:
 - criar, remover ou alterar
 - as relações estão corretas? *Avião/Asa*

Conclusão

- Série de etapas do processo de definição
- Cada etapa pode alterar considerações anteriores
- refazer antigas etapas

Ferramenta

- Deveria ter:
 - Sistema gráfico que suporte notação de projeto
 - *browser*
 - gerência de biblioteca

Estado da Arte

- C++, Smalltalk, Eiffel, Delphi:
 - navegar
 - dependências
 - definir classes via edição
- Especialização é um processo normal
- Generalização é correção no projeto
- reúso
 - conhecer componentes
 - pesquisar na biblioteca

Class Design

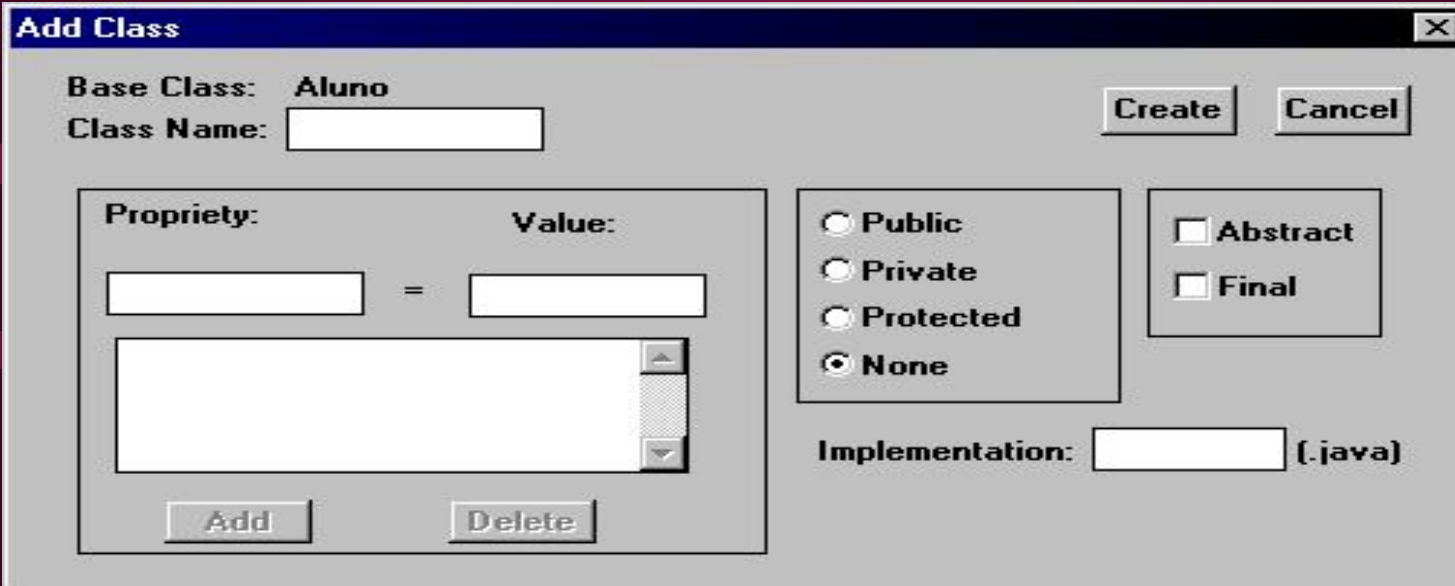
- Notação simplificada de projeto para representar:
 - classes, métodos, asserções
 - propriedades
 - hierarquias
- Recuperação por assinatura ou propriedades
- Navegação pela hierarquia
- Especializações em nível de projeto
- Generalizações em nível de projeto

Class Design...

- Geração de protótipos de módulo
- Manutenção automática dos protótipos
- Edição e compilação das classes
- Incorporação de alterações via edição
- Manipulação de membros e classes

Tela Principal - Class Designer

Inserção de Classe



The image shows a 'Add Class' dialog box with a title bar containing a close button. The dialog is divided into several sections. At the top left, 'Base Class:' is set to 'Aluno', and 'Class Name:' is followed by an empty text box. To the right of these are 'Create' and 'Cancel' buttons. Below the 'Class Name' field is a table with two columns: 'Propriety:' and 'Value:'. The first row of the table has empty input fields for both columns, separated by an equals sign. Below this is a large empty text area with scrollbars. At the bottom of the table are 'Add' and 'Delete' buttons. To the right of the table is a group box containing four radio buttons: 'Public', 'Private', 'Protected', and 'None' (which is selected). To the right of this group box is another group box containing two checkboxes: 'Abstract' and 'Final'. At the bottom right, the 'Implementation:' field is empty, followed by the file extension '(.java)'.

Add Class

Base Class: **Aluno**

Class Name:

Create **Cancel**

Propriety:		Value:
<input type="text"/>	=	<input type="text"/>
<input type="text"/>		

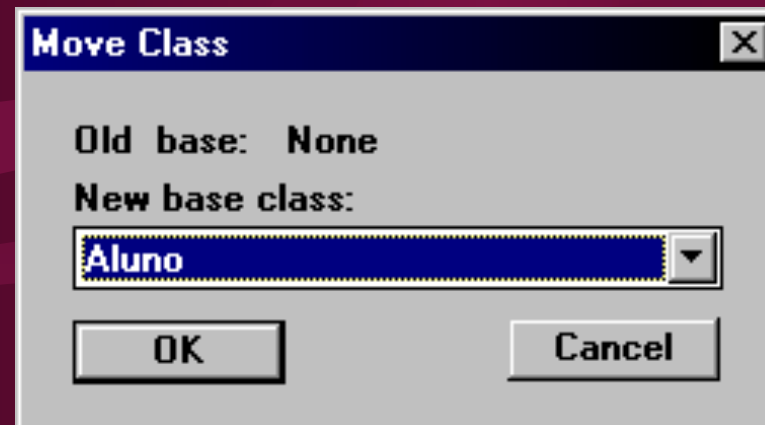
Add **Delete**

☐ **Public**
☐ **Private**
☐ **Protected**
☒ **None**

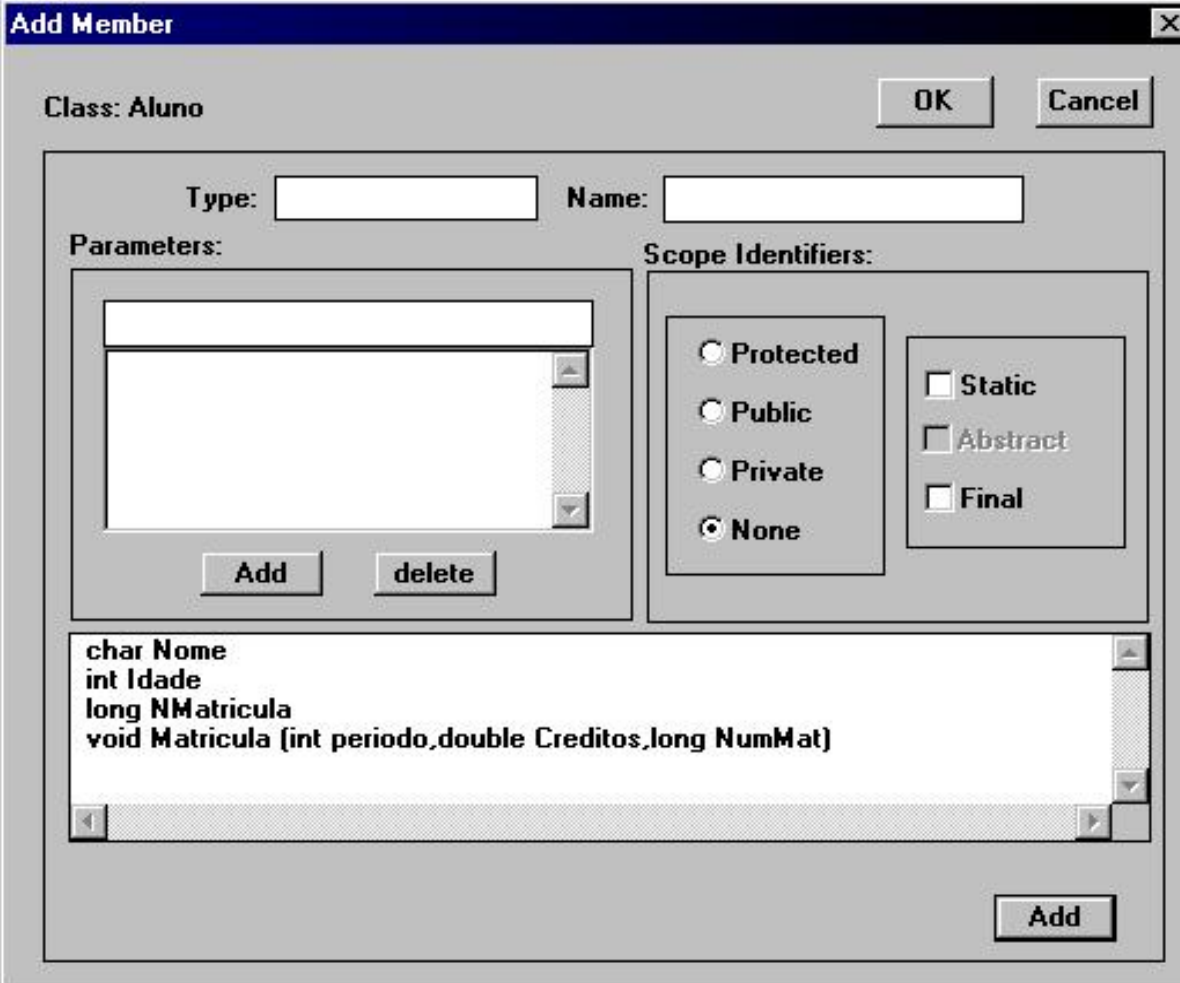
☐ **Abstract**
☐ **Final**

Implementation: **(.java)**

Movimentação de Classe



Inserção de Membros



The image shows a Java IDE 'Add Member' dialog box for the class 'Aluno'. The dialog has a title bar 'Add Member' with a close button. It contains fields for 'Type' and 'Name', a 'Parameters' list with 'Add' and 'delete' buttons, 'Scope Identifiers' with radio buttons for 'Protected', 'Public', 'Private', and 'None' (selected), and checkboxes for 'Static', 'Abstract', and 'Final'. At the bottom is a large text area for the member signature and an 'Add' button.

Add Member

Class: Aluno

OK Cancel

Type: Name:

Parameters:

Add delete

Scope Identifiers:

☐ Protected ☐ Static

☐ Public ☐ Abstract

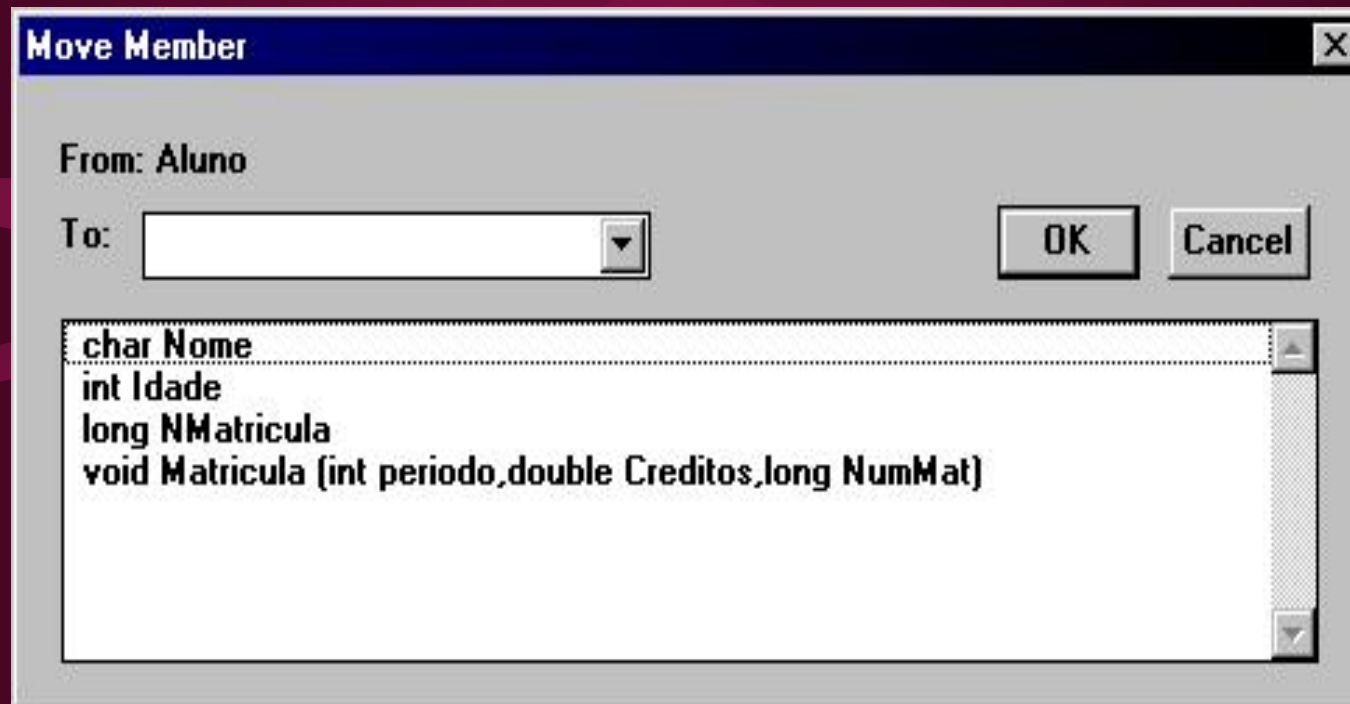
☐ Private ☐ Final

☒ None

char Nome
int Idade
long NMatricula
void Matricula (int periodo,double Creditos,long NumMat)

Add

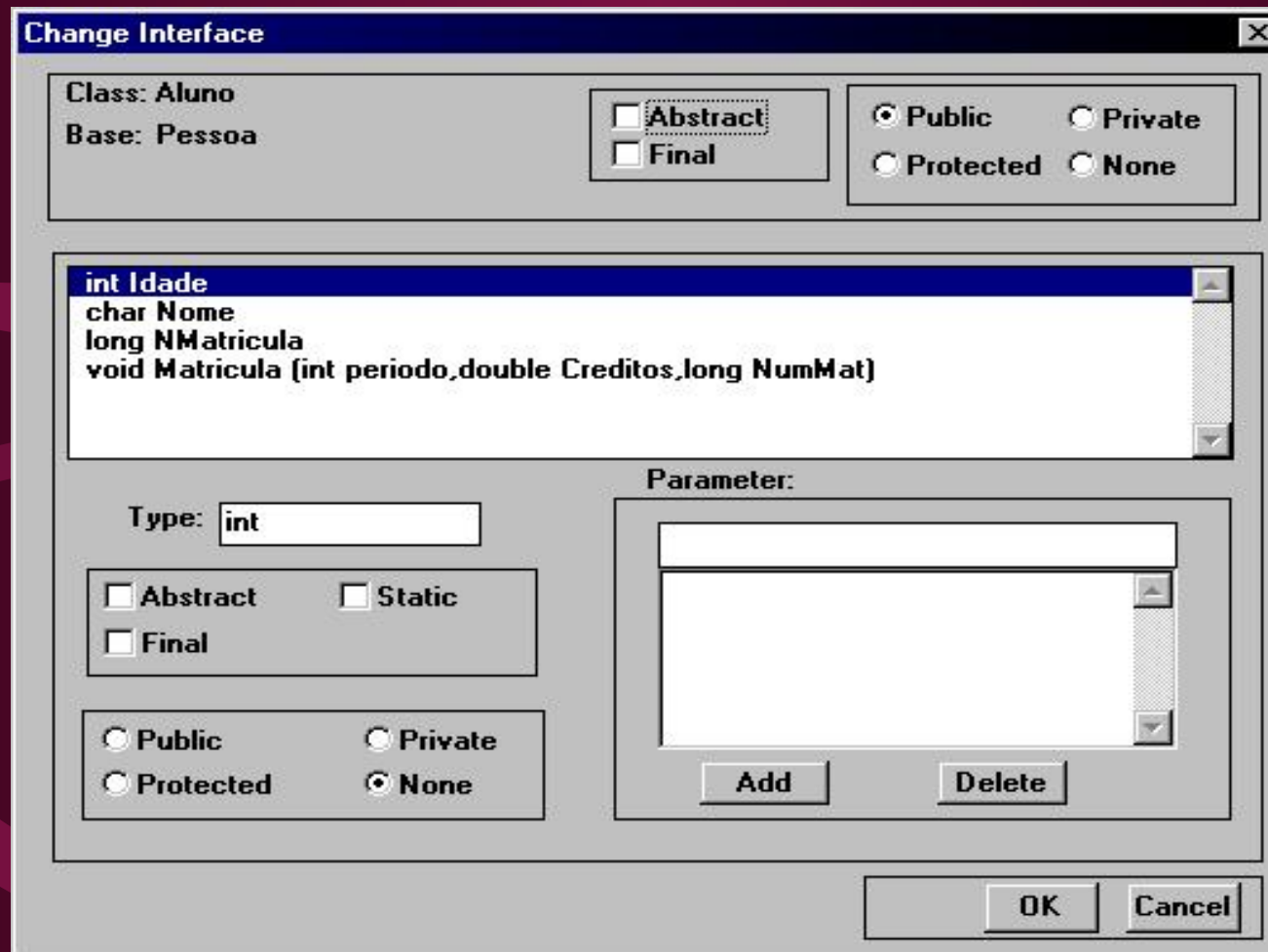
Movimentação de Membros



A dialog box titled "Move Member" with a standard Windows-style title bar (blue with a close button 'X'). The dialog contains a "From:" label followed by the text "Aluno". Below this is a "To:" label followed by an empty text input field with a small downward arrow on the right. To the right of the input field are two buttons: "OK" and "Cancel". At the bottom of the dialog is a large text area with a dotted border, containing the following text:

```
char Nome  
int Idade  
long NMatricula  
void Matricula (int periodo,double Creditos,long NumMat)
```

Alteração da Classe e Membros



The image shows a 'Change Interface' dialog box with a title bar containing a close button. The dialog is divided into several sections. The top section contains 'Class: Aluno' and 'Base: Pessoa'. To the right are checkboxes for 'Abstract' and 'Final', and radio buttons for 'Public' (selected), 'Private', 'Protected', and 'None'. The middle section is a list box containing 'int Idade', 'char Nome', 'long NMatricula', and 'void Matricula (int periodo,double Creditos,long NumMat)'. Below the list box, there is a 'Type:' label followed by a text box containing 'int'. To the right of this is a 'Parameter:' label followed by a large empty text box. Below the 'Type:' section are checkboxes for 'Abstract', 'Static', and 'Final', and radio buttons for 'Public', 'Private', 'Protected', and 'None' (selected). Below the 'Parameter:' section are 'Add' and 'Delete' buttons. At the bottom right are 'OK' and 'Cancel' buttons.

Change Interface

Class: Aluno
Base: Pessoa

☐ Abstract
☐ Final

☒ Public ☐ Private
☐ Protected ☐ None

int Idade
char Nome
long NMatricula
void Matricula (int periodo,double Creditos,long NumMat)

Type: int

☐ Abstract ☐ Static
☐ Final

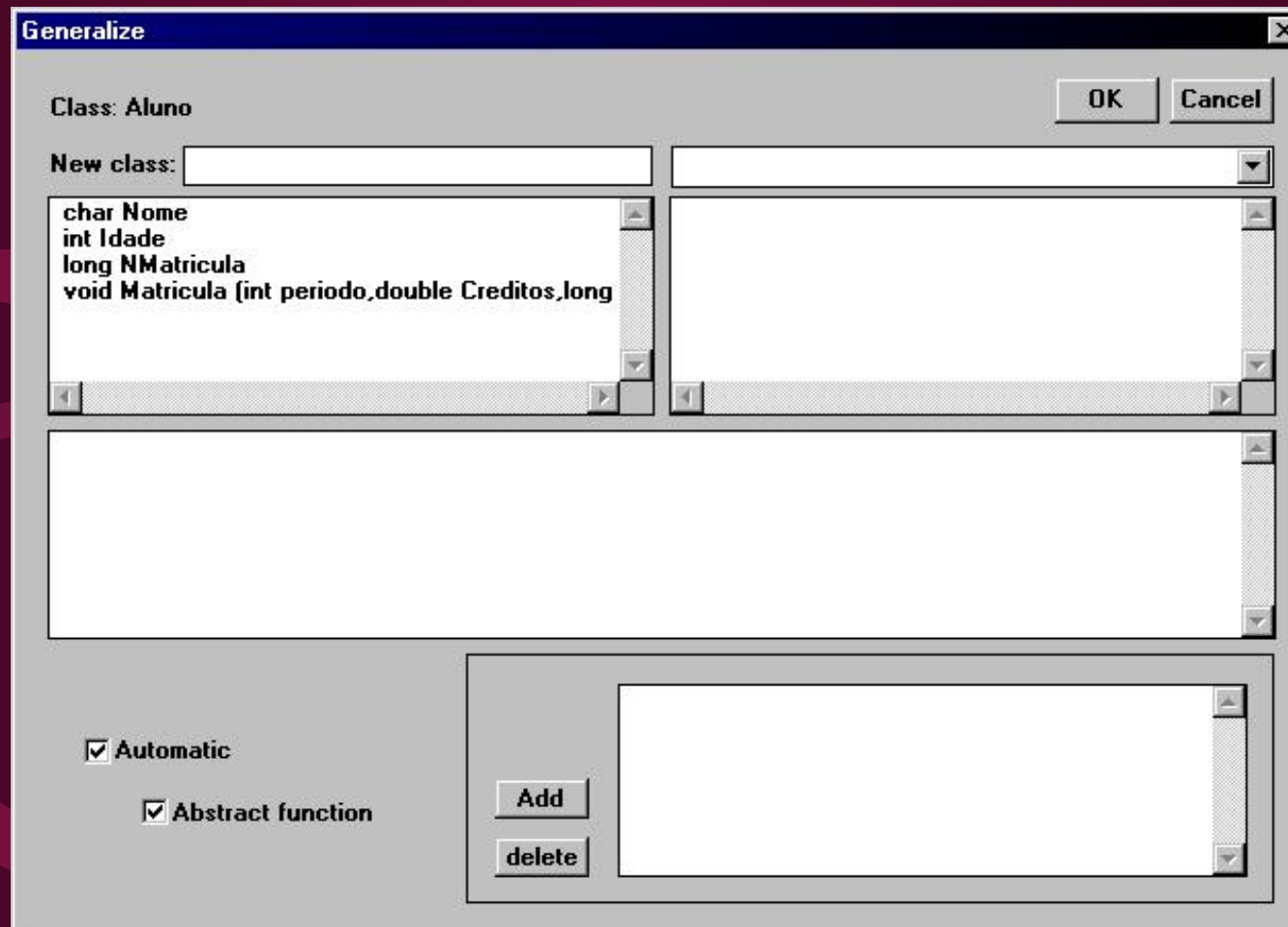
☒ Public ☐ Private
☐ Protected ☒ None

Parameter:

Add Delete

OK Cancel

Generalização



The image shows a 'Generalize' dialog box from a UML modeling software. The dialog has a title bar with a close button. It contains a 'Class: Aluno' label, 'OK' and 'Cancel' buttons, and a 'New class:' label with two empty text boxes. Below these are two large text areas for listing attributes and methods. The left area contains the text: 'char Nome', 'int Idade', 'long NMatricula', and 'void Matricula (int periodo,double Creditos,long'. Below the text areas are two checkboxes: 'Automatic' and 'Abstract function', both of which are checked. At the bottom right, there are 'Add' and 'delete' buttons next to a small text area.

Generalize

Class: Aluno

OK Cancel

New class:

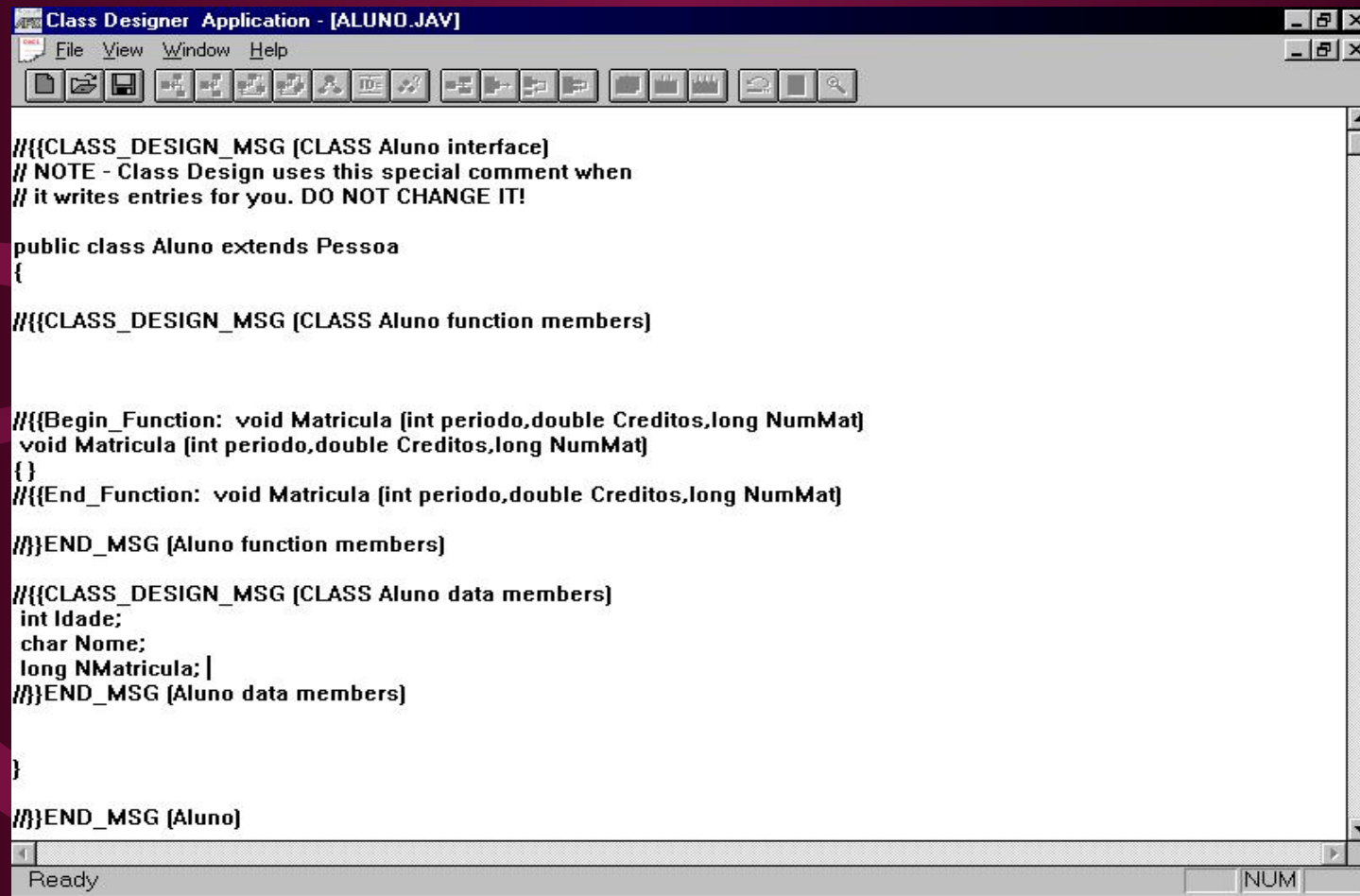
char Nome
int Idade
long NMatricula
void Matricula (int periodo,double Creditos,long

☒ Automatic

☒ Abstract function

Add delete

Edição do Arquivo da Classe



Propriedades

- Pares (**id**, **valor**) (constantes)
- Caracterizar e “direcionar” a implementação
- Não aparecem no código Java
- Não são herdadas por subclasses

Recuperando Classes por Propriedade

- $(p_1, p_2 \dots p_n)$ as propriedades de A
- $(q_1, q_2 \dots q_m)$ as propriedades de B
- Recupera-se B a partir das propriedades definidas em A se:
 - para cada p_i ($1 \leq i \leq n$) existir um
 - q_j ($1 \leq j \leq m$) tal que $p_i = q_j$

Protótipo de Membro

Tupla que identifica um membro da classe

1) Membro de dado: (*Tipo*, *Nome*)

```
class A <T> : C <T> {  
    public:  
        int x;  
        T w;  
        .....  
}
```

int x \rightarrow (int, x)

T w \rightarrow (T, w)

... Protótipo de Membro

2)Membro de função:

(tipo_retorno, nome, lista_param)

```
class A <T> : C <T>    {
```

```
    public:
```

```
        void f (int x);
```

```
        .....
```

```
    }
```

```
void f (int x) → (void, f, { (int, x) } )
```

Protótipo de Classe

Tupla que identifica classe no projeto:

(nome, {*param_gen*}, {*protótipo_membro*})

```
class A <T> : C <T>
{
    .....
    public:
        int x;
        void f (int w);
        .....
}
```

Protótipo = (A, {T}, { (int, x), (void, f, { (int, w) }) })

Assinaturas

SigM: protótipo-membro \rightarrow Assinatura de Membro

Assinatura de Membro:
(tipo_membro_dado) ou
(tipo_retorno, {tipos_param})
onde {tipos_param} ordenada por tipo

```
class A <T> : C <T> {  
    public:  
        int x;  
        void f (int w);  
        .....  
}
```


... Assinaturas

- Membro *int x* tem $P_1 = (\text{int}, x)$ então
 - $\text{SigM}(P_1) = (\text{int})$
- Membro *void f(int w)* tem $P_2 = (\text{void}, f, \{(\text{int}, x)\})$ então
 - $\text{SigM}(P_2) = (\text{void}, \{ (\text{int}) \})$
- A tupla retornada por $\text{SigM} \rightarrow$ **assinatura do membro**

Assinaturas de Classes

SIGC: protótipo_classe \rightarrow Assinatura de Classe

Assinatura de Classe:

({ param_genérico }, { assinatura_membro })

onde {assinatura_membro} ordenada por tipo e valor de retorno

```
class A < T > : C < T > {  
    public:  
        int x;  
        void f (float w, int r);  
}
```

O protótipo da classe A:

$$\mathbf{P}_A = (A, \{T\}, \{ (int, x), (void, f, (float, w), (int, r)) \})$$

... Assinatura de Classes

- $\mathbf{P}_A = (A, \{T\}, \{ (int, x), (void, f, (float, w), (int, r)) \})$
- Logo:
- $SigC(\mathbf{P}_A) = (\{T\}, \{(int), (void, \{(float), (int)\})\})$
- Tupla retornada por $SigC \rightarrow$ **assinatura da classe**

... Equivalência de Classes

- S_A assinatura da classe A
- S_B assinatura da classe B
- A é equivalente (\approx) a B se $S_A = S_B$

Recuperação de Componentes

- $Sel(A) \rightarrow$ subconjunto das assinaturas dos membros de A especificado pelo projetista
- $Membros(B) \rightarrow$ assinaturas dos membros de B
- $Gen(A) \rightarrow \{ \text{parâmetros genéricos} \}$
- S_A assinatura de A
- S_B assinatura de B , **candidata a reúso** :
 - 1. $S_B \approx S_A$ **ou**
 - 2. $S_B \approx S_A$ (A e B **não genéricas**), mas $Membros(B) \subseteq Sel(A)$ **ou**
 - 3. $S_B \approx S_A$ (A e B **genéricas**), mas
 - $Gen(B) = Gen(A)$ e $Membros(B) \subseteq Sel(A)$.

...Recuperação de Componentes

```
class A < T, D:C >  
{ .....  
    Public:  
        int a;  
        float f (int x);  
}
```

```
class B < W:C, K>  
{ .....  
    Public:  
        int c;  
        float k (int x);  
        float g (int w);  
}
```

... Recuperação de Componentes

- $M_1 = \text{int } a;$
- $M_2 = \text{float } f(\text{int } x);$
- $M_3 = \text{int } c;$
- $M_4 = \text{float } k(\text{int } x);$
- $M_5 = \text{float } g(\text{int } w);$
 - e que $\text{Sel}(A) = \{\text{int } a, \text{float } f(\text{int } x)\}$ então por (3) temos:
- $S_B \approx S_A,$
- $\text{Gen}(B) = \text{Gen}(A)$, por exemplo, $\{C, \mathbb{R}\}$ e
- $\text{Members}(B) \subseteq \text{Sel}(A).$

... Recuperação de Componentes

```
class A
{ .....
  public:
    STACK m;
}
```

```
class B
{ .....
  public:
    PILHA c;
}
```


Conclusões

- **Class Design** é uma ferramenta que foi desenvolvida para suporte ao projeto e implementação de classes Ita e Java.
- **Class Design** é uma ferramenta que facilita o desenvolvimento de sistemas.
- Vantagens (*Good* de Eiffel, Class Wizard de C++, *Browser* de Delphi) definir hierarquia e posteriormente efetivar sua implementação.

... Conclusões

- Nos ambientes pesquisados não encontramos uma ferramenta que realmente permitisse o projeto de classes.
- Ferramentas que permitem visualizar as classes da biblioteca, as relações de dependência, os atributos e métodos.

Limitações e Trabalhos Futuros

- A impossibilidade de se editar classes com mais de 64 kbytes.
- A impossibilidade de se trocar informações entre projetos distintos.
- A ausência de um mecanismo que permita controlar ou visualizar a execução de objetos.
- A impossibilidade de se operar diretamente com os membros na janela principal.
- A interface adota as sintaxes Ita e Java para textos de programas, com cabeçalhos de funções e a geração de protótipos, não sendo por isso totalmente independente de linguagem.