Static Metaprogramming In C++

K. CZARNECKI e U. EISENECKER

Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística

Edson Ricardo Amboni (amboni@inf.ufsc.br)
John Cleber Jaraceski (john@inf.ufsc.br)
Ronivaldo Fernando Moretti (moretti@inf.ufsc.br)

Metaprogramming

"A arte de programar programas que lêem, transformam ou escrevem outros programas"

François-René Rideau, 1999

- Compiladores
- Interpretadores
- Depuradores

Classificação

Entradas $\xrightarrow{\text{metaprograma}}$ Saídas

- Interpretador
- *Inspector*
- Tradutor
- Compilador
- *Splitter*

- Metainterpretador
- Analisador de diferenças
- *Walker*
- Metacompilador

Static Metaprogramming in C++

Metaprograma que executa antes do *load-time* do programa manipulado.

- Pré-processador
- Templates
  - Standard Template Library

Características

- Aninhamento de conceitos
- Algoritmos especializados
- Sub-linguagem da própria linguagem
- Eliminação de *overhead*
- Compilador como máquina virtual
- Tecnologia padronizada

Classificação

- *Metafunction*
  - Estrutura que manipula dados estáticos de um programa que ainda não foi gerado
- *Metainformation* ⇒ *Trait templates*
  - Estrutura que armazena informações sobre outros elementos (tipos)
- *Metaprogram* ⇒ *Generator*
  - *Metafunctions* que fazem geração de código

Pré-processador [1]

Programa que processa um código fonte (eventualmente o altera) antes deste ser submetido ao compilador.

Pré-processador 2

```
#include <iostream.h>

#define SOUnix

void main() {
  #ifdef SOUnix
    cout << "SO Unix" << endl;
  #else
    cout << "SO Not Unix - MS-Windows?" << endl;
  #endif
}
```

Templates [1]

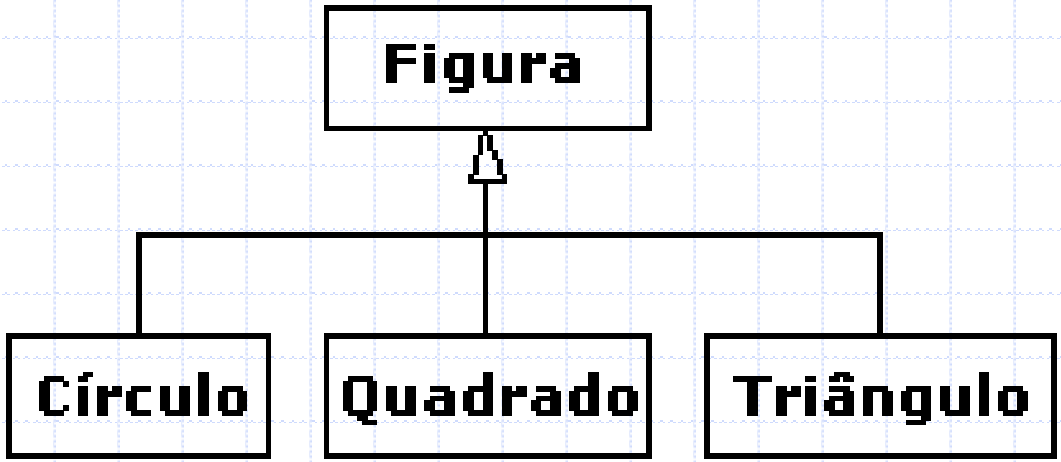
"Os Templates presentes em C++ fornecem uma forma para reusar código fonte."

Bruce Eckel, 2.000

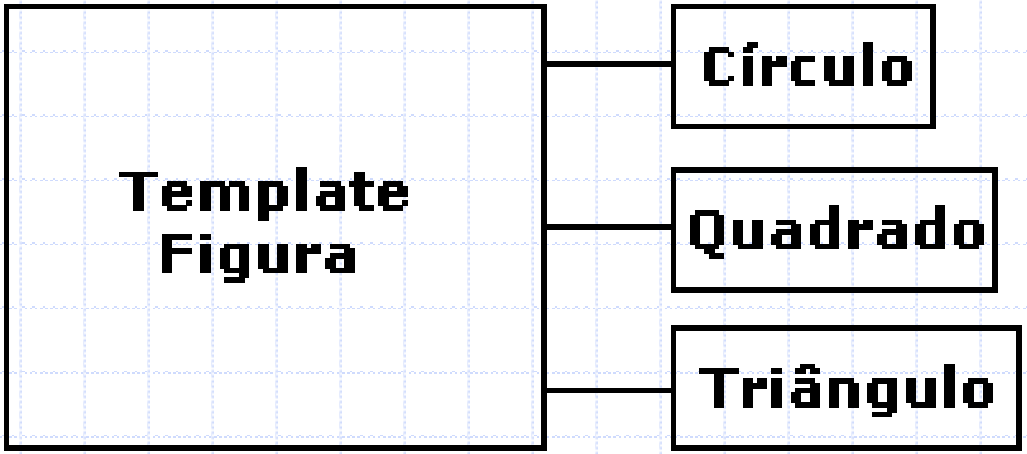
- Reutilização de código fonte ao invés de código objeto
- Operam sobre números e tipos de dados
- Algoritmos de recursão e especialização

Templates 2

Reuso
Código
Objeto



Reuso
Código
Fonte



Declaração e Utilização

Parâmetro do Template

Cabeçalho do Template

```
template<class T>
T max(T a, T b) {
  return (a > b) ? a : b;
};
```

Corpo do Template

```
double x = 10, y, d_max;

cin >> y;

d_max = max(x, y);
```

Uso do Template

Como o compilador funciona

```
d_max = max(x, y);
```

```
template<class T>
T max(T a, T b) {
  return (a > b) ? a : b;
};
```

Instância do Template

```
double max_double(double a, double b) {
  return (a > b) ? a : b;
};
```

```
d_max = max_double(x, y);
```

Recursividade 1

```
#include <iostream.h>

template<int N>
struct fatorial {
  enum { RET = N * fatorial<N - 1>::RET };
};

template<>
struct fatorial<0> {
  enum { RET = 1 };
};

void main() {
  cout << fatorial<7>::RET << endl;
}
```

Recursividade 2

```
cout << fatorial<2>::RET << endl;
```

```
struct fatorial<2> {
  enum { RET = 2 * fatorial<1>::RET };
};
```

```
struct fatorial<1> {
  enum { RET = 1 * fatorial<0>::RET };
};
```

1

```
struct fatorial<1> {
  enum { RET = 1 * 1 };
};
```

```
struct fatorial<2> {
  enum { RET = 2 * 1 };
};
```

Recursividade 3

```
_main:
  pushl %ebp
  movl %esp,%ebp
  subl $8,%esp
  call ___main
  addl $-8,%esp
  pushl $_endl__FR7ostream
  addl $-8,%esp
  pushl $5040
  pushl $_cout
```

```
_main:
  pushl %ebp
  movl %esp,%ebp
  subl $8,%esp
  call ___main
  addl $-8,%esp
  pushl $_endl__FR7ostream
  addl $-8,%esp
  addl $-12,%esp
  pushl $7
  call _fatorial__Fi
  addl $16,%esp
  movl %eax,%eax
  pushl %eax
  pushl $_cout
```

Combinação

```
#include <iostream.h>
#include “fatorial.h”

template<int n, int p>
struct combinacao {
  enum { RET = fatorial<n>::RET /
               ( fatorial<p>::RET * fatorial<n - p>::RET )
  };
};


void main() {
  cout << combinacao<2, 4>::RET << endl;
}
```

Especialização

```
#include <iostream.h>

template<int n, int p>
struct potencia {
  enum { RET =
     (p > 0) ? potencia<n, ((p > 0) ? (p - 1) : 0)>::RET * n : 1
  };
};

template<int n>
struct potencia<n, 0> {
  enum { RET = 1 };
};

void main() {
  cout << potencia<2, 3>::RET << endl;
}
```

Estruturas de controle: IF

```
#include <iostream.h>

template<bool condicao, class ThenType, class ElseType>
struct IF {
  typedef ThenType RET;
};

template<class ThenType, class ElseType>
struct IF<false, ThenType, ElseType> {
  typedef ElseType RET;
};

void main() {
  IF< (1 > 2), int, double >::RET valor = 1.1;

  cout << valor << endl;
}
```

Estruturas de controle: WHILE [1]

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

template<int i>
struct meuComando {
  enum { n = i };
  static void executar() { cout << i << endl; };
  typedef meuComando<n + 1> proximo;
};

struct minhaCondicao {
  template<class Comando>
  struct Teste {
    enum { avaliacao = ( Comando::n <= 10 ) };
  };
};

void main() {
  WHILE< minhaCondicao , meuComando<1> >::EXECUTAR();
}
```

Estruturas de controle: WHILE 2

```
// Arquivo: while.h

struct Parar {
  static void EXECUTAR() {};
  static void executar() {};
};

template <class Condicao, class Comando>
struct WHILE {
  static void EXECUTAR() {
    IF< ( Condicao::Teste<Comando>::avaliacao != 0 ) ,
            Comando, Parar
      >::RET::executar();

    IF< ( Condicao::Teste<Comando>::avaliacao != 0 ) ,
            WHILE<Condicao, Comando::proximo>, Parar
      >::RET::EXECUTAR();
  }
};
```

... In Object Pascal

- Pré-processador
- “Templates”

Pré-processador

```
Program so;

{$APPTYPE CONSOLE}
{$DEFINE SOUnix}

Begin
  {$IFDEF SOUnix}
    WriteLn( 'SO Unix' );
  {$ELSE}
    WriteLn( 'SO Not Unix – MS-Windows?' );
  {$ENDIF}
End.
```

“Templates” 1

```
_VECTOR_CLASS_= class

private
 FArray : array of _VECTOR_DATA_TYPE_;
protected
 function  GetLength():Integer;
 procedure SetLength(const aLength:Integer);
 function  GetItems(const aIndex:Integer):_VECTOR_DATA_TYPE_;
 procedure SetItems(const aIndex:Integer;
                    const aValue:_VECTOR_DATA_TYPE_);
public
 function Clear():_VECTOR_CLASS_;
 function Extend(const aDelta:Word=1):_VECTOR_CLASS_;
 function Contract(const aDelta:Word=1):_VECTOR_CLASS_;
 property Length:Integer read GetLength write SetLength;
 property Items[const aIndex:Integer]:_VECTOR_DATA_TYPE_
          read GetItems write SetItems; default;
 constructor Create(const aLength:Integer);
end;
```

“Templates” 2

```
constructor _VECTOR_CLASS_.Create(const aLength : Integer);
begin
 inherited Create;
 SetLength (aLength);
end;

...

function _VECTOR_CLASS_.Contract(const aDelta : Word) :
   _VECTOR_CLASS_;
begin
 System.SetLength (FArray, System.Length (FArray) - aDelta);
 Result := Self;
end;
```

“Templates” 3

```
unit StrVector;
interface
uses Classes;
type _VECTOR_DATA_TYPE_ = String;
     {$INCLUDE TemplateVectorInterface}
     TStringVector = _VECTOR_CLASS_;
implementation
{$INCLUDE TemplateVectorImplementation}
end.

unit FltVector;
interface
uses Classes;
type _VECTOR_DATA_TYPE_ = Double;
     {$INCLUDE TemplateVectorInterface}
     TFloatVector = _VECTOR_CLASS_;
implementation
{$INCLUDE TemplateVectorImplementation}
end.
```

“Templates” 4

```
program VectorTest;
 {$DEFINE FloatVector}
 {$APPTYPE CONSOLE}
uses
 SysUtils,
 {$IFDEF StrVector} StrVector in 'StringVector.pas'; {$ENDIF}
 {$IFDEF FloatVector} FltVector in 'FltVector.pas'; {$ENDIF}
var Vector : _VECTOR_CLASS_;
begin
 {$IFDEF StringVector}
  Vector := TStringVector.Create( 1 );
  Vector[ 0 ] := 'String';
  Writeln( 'Estou trabalhando com um vetor de strings!' );
  Writeln( 'Vetor[0] = ''' + Vector[ 0 ] + ''''' ); {$ENDIF}
 {$IFDEF FloatVector}
  Vector := TFloatVector.Create( 1 );
  Vector[ 0 ] := 0.017;
  Writeln( 'Estou trabalhando com um vetor de floats!' );
  Writeln( 'Vetor[0] = ' + FloatToStr( Vector[ 0 ] ) ); {$ENDIF}
end.
```

“Templates” 5

```
program VectorTest;
 {$DEFINE StringVector}
 {$APPTYPE CONSOLE}
uses
 SysUtils,
 {$IFDEF StrVector} StrVector in 'StringVector.pas'; {$ENDIF}
 {$IFDEF FloatVector} FltVector in 'FltVector.pas'; {$ENDIF}
var Vector : _VECTOR_CLASS_;
begin
 {$IFDEF StringVector}
  Vector := TStringVector.Create( 1 );
  Vector[ 0 ] := 'String';
  Writeln( 'Estou trabalhando com um vetor de strings!' );
  Writeln( 'Vetor[0] = ''' + Vector[ 0 ] + ''''' ); {$ENDIF}
 {$IFDEF FloatVector}
  Vector := TFloatVector.Create( 1 );
  Vector[ 0 ] := 0.017;
  Writeln( 'Estou trabalhando com um vetor de floats!' );
  Writeln( 'Vetor[0] = ' + FloatToStr( Vector[ 0 ] ) ); {$ENDIF}
end.
```

Generative Programming

- Engenharia de domínios
  - Família de sistemas
- Configuração de bases de conhecimento
  - Mapeamento de domínios e Regras
- Geradores de código

Características

- Parametrização
- Modelagem de dependências
- Eliminação de *overhead*
- Separação de domínios
- Separação de interesses

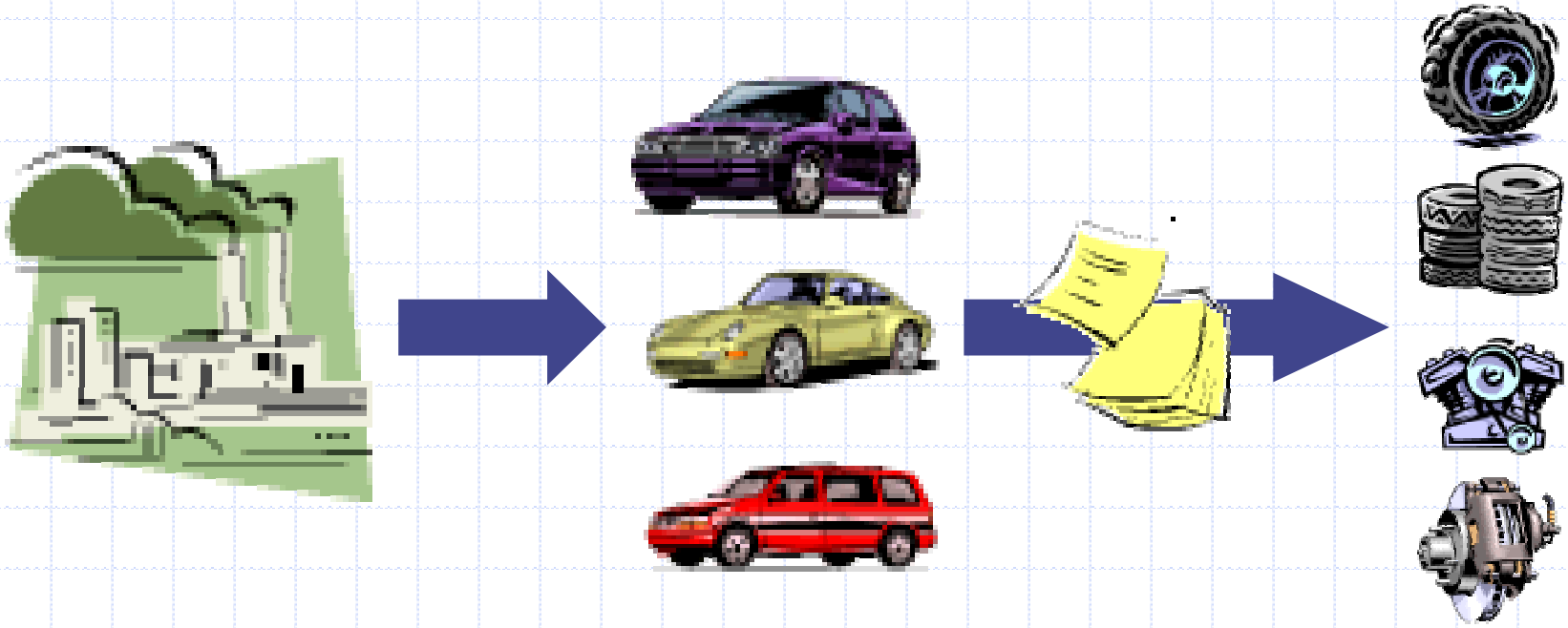
Paradigmas relacionados

- *Generic Programming*
  - Reuso por parametrização
- *Aspect-Oriented Programming*
  - Separação de interesses
- *Domain-Specific Language*
  - Linguagem de alto nível de abstração

Linha de Produção de Carros

- Domínio do problema
- Domínio da solução
  - Configuração base de conhecimento
- Implementação dos componentes

Analisando o problema



Definindo modelos de carros

Modelos de Carros	Básico	Esportivo
Motor		
Gasolina	●	
Álcool	○	●
Roda		
Normal	●	●
Esportiva		○

● **equipamento padrão**

○ **equipamento alternativo**

Utilizando gerador de carros

```
#include <iostream.h>
#include “ger_carro.h”

void main() {
  typedef GERADOR_CARRO<Esportivo<> >::Carro Modelo_A;
  typedef GERADOR_CARRO<Esportivo<esportiva> >::Carro Modelo_B;
  typedef GERADOR_CARRO<Basico<> >::Carro Modelo_C;
  typedef GERADOR_CARRO<Basico<alcool> >::Carro Modelo_D;

  Modelo_A meu_carro1;
  Modelo_B meu_carro2;
  Modelo_C meu_carro3;
  Modelo_D meu_carro4;
}
```

Implementando a solução 1

```
enum Roda { roda_normal, roda_esportiva };
enum Modelo { modelo_basico, modelo_esportivo };
enum Motor { motor_gasolina, motor_alcool };


template<int roda_ = roda_normal>
struct Esportivo {
  enum { motor = motor_gasolina,
         roda = roda_,
         modelo = modelo_esportivo };
};

template<int motor_ = gasolina>
struct Basico {
  enum { motor = motor_,
         roda = roda_normal,
         modelo = modelo_basico };
};
```

Implementando a solução 2

```
template<class Config>
struct MotorGasolina {
  typedef Config_ Config;

  MotorGasolina() {
    cout << "Motor Gasolina ";
  }
};

template<class Config>
struct MotorAlcool{
  typedef Config_ Config;

  MotorAlcool() {
    cout << "Motor Alcool ";
  }
};
```

Implementando a solução 3

```
template<class Config_>
struct RodaNormal {
  typedef Config_ Config;
  enum { aro = Config::aro };

  RodaAroNormal() {
    cout << "Roda Normal Aro " << aro << ": " ;
  }
};

template<class Config_>
struct RodaEsportiva{
  typedef Config_ Config;
  enum { aro = Config_::aro };

  RodaAroEsportivo() {
    cout << "Roda Esportiva Aro " << aro << ": ";
  }
};
```

Implementando a solução 4

```
template<class Motor_>
struct Carro {
  typedef typename Motor_::Config Config;

  Motor_ Motor;

  IF< (Config::aro == 15),
      RodaAroEsportivo<Config>,
      RodaAroNormal<Config>
    >::RET Roda;

  Carro() {
    cout << "Carro: " << endl << endl;
  }
};
```

Implementando a solução 5

```
template<class Modelo = Basico<> >
struct GERADOR_CARRO {
  typedef GERADOR_CARRO<Modelo> Gerador;
  enum { aro = (Modelo::roda == roda_normal) ? 12 : 15 };

  typedef IF< (Modelo::motor == motor_gasolina),
              MotorGasolina<Gerador>, MotorAlcool<Gerador>
            >::RET Motor_;

  typedef IF< (Modelo::roda == roda_normal),
              RodaAroNormal<Gerador>, RodaAroEsportivo<Gerador>
            >::RET Roda_;

  typedef Carro<Motor_> RET;

  struct Config {
    typedef Motor_ Motor; typedef Roda_ Roda;
    enum { aro = aro_ , modelo = modelo_ };
    typedef RET Carro;
  };
};
```

Deficiências

- Despadronização de compiladores
- Abstrações incompletas
- Complexidade de implementação
- Considerações econômicas
- Propriedade intelectual

Um Caso de Metaprogramming

Através de tabelas do Banco de Dados, gerar classes básicas do Domínio do Problema.

Características

- Template em XSL
- Comunicação do Banco de Dados com pré-compiladores através de XML
- Geração código fonte Java
  - Classes que representam as tabelas do banco de dados

Banco de Dados

- SERIE
  - CODIGO
  - NOME
  - GRAU
  - Indices
- CURRICULO
  - CODIGO
  - COD_SERIE
  - COD_DISC
  - Indices
- DISCIPLINA
  - CODIGO
  - NOME
  - CONTEUDO
  - AULASHORA
  - Indices
- TURMA
  - CODIGO
  - NOME
  - COD_SERIE
  - FAZERHORARIO
  - PERIODO
  - Indices
- AULA
  - CODIGO
  - COD_DISC
  - COD_TURMA
  - COD_PROF
  - COD_HORARIO
  - PREENCHIDA
  - Indices
- DISCIPLINAPROFESSOR
  - CODIGO
  - COD_PROF
  - COD_DISC
  - Indices
- AGENDASEMANAL
  - CODIGO
  - COD_HORARIO
  - COD_PROFESSOR
  - ESTADO
  - Indices
- HORARIO
  - CODIGO
  - DIA
  - ORDEM
  - PERIODO
  - Indices
- ESPECIFICACAOHORARIO
  - CODIGO
  - HORAINICIO
  - HORAFIM
  - HORAINTERVALO
  - AULASPORDIA
  - DIASAULA
  - Indices
- DIASSEMANA
  - CODIGO
  - NOME
  - Indices

Template em XSL

```
public class <xsl:apply-templates select="classe/nome"/> {
  <xsl:for-each select="/classe/atributos/atributo">
  protected <xsl:value-of select="tipo"/> <xsl:text>  </xsl:text> <xsl:value-of select="nome"/>
  </xsl:for-each>
  public <xsl:apply-templates select="classe/nome"/>() {
  }
  public <xsl:apply-templates select="classe/nome"/>(<xsl:for-each select="/classe/atributos/at
      <xsl:for-each select="/classe/atributos/atributo">this.<xsl:value-of select="nome"/> = <x
      </xsl:for-each>
  }  <xsl:for-each select="classe/metodos/metodo">
  public <xsl:value-of select="retorno"/> <xsl:text>  </xsl:text> <xsl:value-of select="nomecom
  <xsl:if test="retorno='void'">
      this.<xsl:value-of select="atributo"/> = <xsl:value-of select="nomeparametro"/>;
  }
  </xsl:if>
  <xsl:if test="retorno!='void'">
      return <xsl:value-of select="atributo"/>;
  }
  </xsl:if>
  </xsl:for-each>
}
  </xsl:template>
```

XML de Comunicação

C:\Meus documentos\Arquivos Java\metaprograming2\codgerado\Diassema...

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites Media

Address C:\Meus documentos\Arquivos Java\metaprograming2\codgerado\Diassemana.x Go

```
  <?xml version="1.0" ?>
- <classe>
    <nome>Diassemana</nome>
  - <atributos>
    - <atributo>
        <nome>codigo</nome>
        <tipo>int</tipo>
        <ultimo>false</ultimo>
      </atributo>
    - <atributo>
        <nome>nome</nome>
        <tipo>String</tipo>
        <ultimo>true</ultimo>
      </atributo>
    </atributos>
  + <metodos>
  </classe>
```

My Computer

Código Java Gerado

```
package codgerado;

public class Agendasemanal {

  protected int  codigo ;

  protected int  cod_horario ;

  protected int  cod_professor ;

  protected char  estado ;

  public Agendasemanal() {
  }
  public Agendasemanal(int codigo_ , int cod_horario_ , int cod_professor_ , cha
      this.codigo = codigo_;
      this.cod_horario = cod_horario_;
      this.cod_professor = cod_professor_;
      this.estado = estado ;
```

Referências Bibliográficas

- R. Assenov. Templates in Object Pascal, Borland Community, 2.001, http://community.borland.com/article/print/0,1410,27603,00.html
- K. Czarnecki e U. Eisenecker. Generative Programming: Methods, Techniques, and Applications, Addison-Wesley, 1.999, pg. 243-269
- K. Czarnecki e U. Eisenecker. Components and Generative Programming, Proceedings of the European Software Engineering Conference, 1.999, http://www.prakinf.tu-ilmenau.de/~czarn/eses99/
- K. Czarnecki, U. Eisenecker, R. Glueck e D. Vandevoorde. Generative Programming and Active Libraries, Proceedings of the Dagstuhl-Seminar on Generic Programming Conference, 1.998, http://osl.iu.edu/~tveldhui/papers/dagstuhl1998/dagstuhl.html

Referências Bibliográficas

- K. Czarnecki, U. Eisenecker e P. Steyaert. Beyond Objects: Generative Programming, ECCOP'97, 1.997, http://citeseer.nj.nec.com/408302.htm
- S. Downey. Template Metaprogramming, Borland Community, 1.999, http://community.borland.com/article/print/0,1772,10526,00.html
- B. Eckel. Thinking in C++: Introduction to Standard C++, Prentice Hall, Volume 1, 2.000, http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html
- F. Rideau. Metaprogramming and Free Availability of Sources. Conference Autour du Libre, 1.999, http://fare.tunes.org/articles/ll99/index.en.html

Dúvidas - Questionamentos