Universidade Tecnológica Federal do Paraná.
Departamento Acadêmico de Eletrônica
prof. Robinson Vida Noronha (vida@utfpr.edu.br) e Bruno Chang

Templates e Pilhas

1. Introdução

"Templates" ou "Generics" é considerado uma forma de polimorfismo [1]. É uma forma de programar onde o tipo do dado não é considerado. Programa-se para um tipo de dado indefinido. O foco do programador deverá estar no algoritmo e não no dado. O algoritmo gerado deverá ser capaz de ser executado para qualquer tipo de dado.

Exemplo de seu emprego: Não seria interessante a existência de uma função "max" que retornasse o maior valor de dois números, independentemente do tipo do número? Se esse é o tipo de questão que precisa ser respondida, então utilizar "templates" é uma opção de solução.

Uma outra aplicação bem interessante de "templates" é para a definição de Pilhas, Listas, Filas, Árvores e Grafos. Neste material, apenas Pilhas serão analisadas. Nas Figura 1a) e 1b), a seguir, dos exemplos de pilhas. Na Figura 1a), uma pilha de moedas de chocolate. Na Figura 1b), uma pilha de pratos.

a)

b)

Figura 1 – Exemplos de Pilhas

Destaca-se que nas Figuras 1a) e 1b) o tipo de elemento que está na pilha não é o mesmo. Na Figura 1a) moedas de chocolate, na Figura 1b) pratos. Mesmo sendo elementos distintos, todos eles possuem a característica de estarem empilhados. Nesse contexto, poderíamos imaginar a existência de um "template" que represente uma pilha de elementos. Esse "template" se bem definido, deveria ser capaz de empilhar e desempilhar pratos, livros, jornais, moedas de chocolate, entre outros elementos.

Nas próximas seções, apresenta-se o conceito de "templates" com implementações nas linguagens Java e C++. Na Seção 2, "templates" de função. Na Seção 3, apresenta-se como é a representação de Classes Templates na linguagem UML. Na Seção 4, por sua vez, representa-se o conceito de pilhas por meio de "templates".

2. Templates de Função

O Quadro 1, a seguir, apresenta um exemplo de implementação da função "template" **max** na linguagem C++. Essa função deverá retornar o maior valor entre dois números quaisquer. Esses números podem ser do tipo inteiro, double, float ou simplesmente char. Observe nesse quadro que:

* Há a necessidade de iniciar o "template" com a instrução **template <class ALIAS>**.
  Nesse exemplo, o tipo do dado ou álias, foi definido apenas por "T" .
* Qualquer referência a esse tipo de dado referenciado por "T", deverá ser feita fazendo uma chamada a esse álias.

Quadro 1 - Exemplo de função Template em C++

```
#include <iostream>

template <class T>
T max(T left, T right)
{  if (left < right)  return right;
   else return left; }

int main()
{     std::cout << max(3,4) << std::endl;
      std::cout << max(4.5, 6.7) << std::endl;
      std::cout << max ('a', 'c') << std::endl;

      return 0;
}
```

O Quadro 2, por sua vez, apresenta um exemplo de implementação dessa função "template" na linguagem Java. Observe que:

*Há a necessidade de inicar a declaração do template com a instrução **<ALIAS> assinaturaDaFuncao( )**.
Nesse exemplo, o tipo do dado foi definido apenas por "T" .

*Qualquer referencia a esse tipo de dado deverá ser feita fazendo uma chamada ao álias.

Quadro 2 - Exemplo de função Template em Java

```
public class Main {

      public static<N> N  max(N left, N right)
      {
        if (left.hashCode() > right.hashCode())
            return left;
        else
             return right;
      }

      public static void main(String[] args) {
            System.out.println(">>" + max(3,4) );
            System.out.println(">>" + max(9.8, 14.0) );
            System.out.println(">>" + max('a','c') );
      }
}
```

Nas linguagens C++ e Java o nome do tipo do dado, ou álias, não possui significado associado algum. Por exemplo, em C++ e em Java, esse "template", poderia também ser definido da seguinte maneira, mudando-se o álias de "T" para "desconhecida":

```
//CODIGO CPP
template <class desconhecida>
desconhecida max(desconhecida left, desconhecida right)
{ if (left < right)  return right; else return left; }
```

```
// CODIGO JAVA
public static <desconhecida>
  desconhecida max(desconhecida left, desconhecida right)
  {  if (left.hashCode() > right.hashCode()) return left;
      else  return right;
  }
```

Apesar dessa liberdade na definição, [2] recomenda que a definição do tipo genérico seja feita iniciando-se com caracteres maiúsculos e que os seguintes caracteres sejam utilizados de acordo com o seu significado:

* K : se for chave
* N : se for número
* T : se for um tipo
* V : se for um valor

3 Representação UML de Templates

A UML define uma maneira própria para representar graficamente um "template". Na representação gráfica da classe deve-se adicionar um retângulo com o álias do template. A Figura 2, a seguir, ilustra esse representação. Esse diagrama informa que a classe deve ser definida como template e haverá um atributo denominado por "Dado" que armazenará esse tipo de dado.

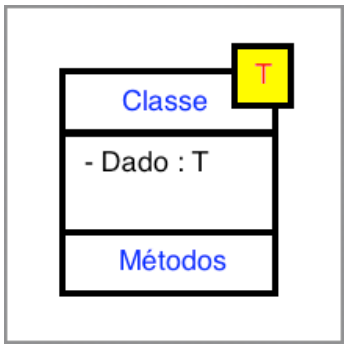

Figura 2 - Representação em UML de "templates" em Classe

4 Uso do Templates para implementar uma Pilha.

Na Figura 3, a seguir, o Diagrama de Classes de um template que permite representar uma pilha de elementos.

Na seção 4.1, esse diagrama é implementado em C++ e um pequeno exemplo é definido para testar as classes. Essas implementações e definições estão definidas nos Quadros 4, 5, 6 e 7.

Na seção 4.2, o mesmo diagrama apresentado na Figura 3 é implementado na linguagem Java. Além disso, a seção 4.2 ainda apresenta um pequeno exemplo de uso desse template.

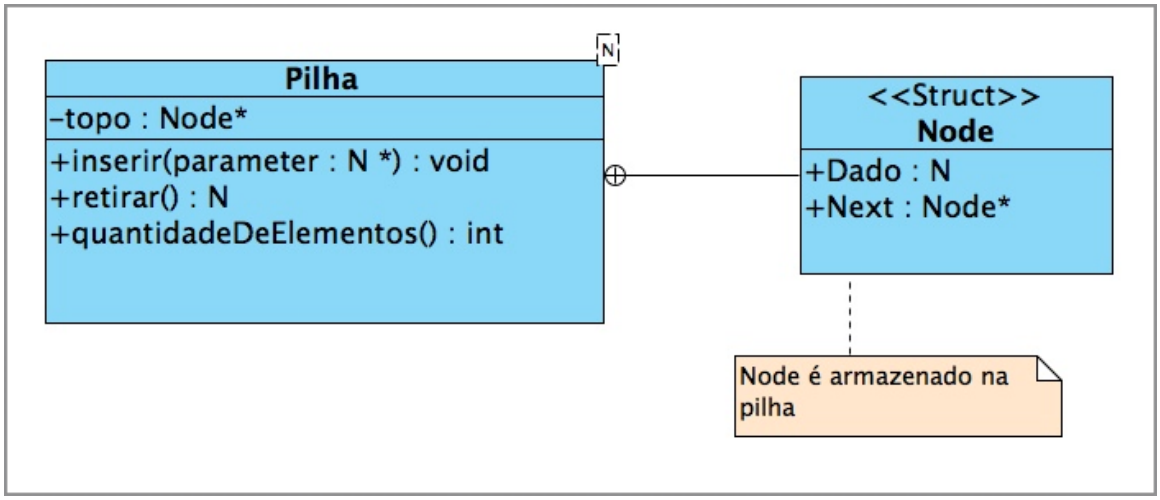

Figura 3 – Diagrama UML de uma Pilha Genérica

4.1 Exemplo de Implementação em C++ do Diagrama da Figura 3

Nos Quadros 4, 5, 6 e 7, a seguir, um exemplo de template que implementa uma pilha que permite armazenar qualquer tipo de elemento.

Quadro 4 - Exemplo de Template que implementa a Pilha Genérica ilustrada na Figura 2 - C++

```
template <class N>
class Pilha
{
   private:
        struct Node
        {   N *dado;
            Node *next;
        } ;
        Node *topo;
   public:
           Pilha(N *v) { topo=NULL; inserir(v);}
           Pilha(){ topo = NULL; }

           void inserir(  N *v )
           {
               struct Node *novo = (struct Node*)
                         malloc(sizeof(struct Node));

               novo->dado = v;
               novo->next = topo;
               topo = novo;
            }

           N giveMeTop() { return *topo->dado; }

           N retirar()
           {
             if (this->quantidadeDeElementos() > 0)
             {
                 N *saida = topo->dado;
                 topo = topo->next;
                 return *saida;
             }
             else {
                    std::cerr << "Pilha Vazia! " << '\n';
                    throw std::out_of_range ("Pilha Vazia");
                   }
           }

           int quantidadeDeElementos()
           {
             Node* navegador = topo;
             int quantidade = 0;
             while (navegador != NULL)
             {quantidade ++;
              navegador = navegador->next;
             }
             return quantidade;
           }};
```

Universidade Tecnológica Federal do Paraná.
Departamento Acadêmico de Eletrônica
prof. Robinson Vida Noronha (vida@utfpr.edu.br) e Bruno Chang

Quadro 5 - Exemplo de Uso do Template Pilha - C++

```
#include <iostream>
#include <stdexcept>
#include "Ponto2D.h"

/* template <class N>
   class Pilha
   { .... vide o código do Quadro 4  */

int main()
{
   /* primeiro exemplo: Pilha de Strings */
    std::string t1 = "1 - oi";
    std::string t2 = "2 - ola";
    std::string t3 = "3 - como vai";

    Pilha<std::string> pilha;
    pilha.inserir(&t1);
    pilha.inserir(&t2);
    pilha.inserir(&t3);

    std::string retirado;

    while( pilha.quantidadeDeElementos() > 0)
    { retirado = pilha.retirar();
      std::cout << retirado << std::endl;}

    /* segundo exemplo da Pilha: Pilha de Instancias de Classe */
    Pilha<Ponto2D> pilhaDePontos;

    Ponto2D *p1 = new Ponto2D(); p1->moverOPonto(3,5);
    Ponto2D *p2 = new Ponto2D(); p2->moverOPonto(7,8);
    Ponto2D *p3 = new Ponto2D(); p3->moverOPonto(12,34);

    Ponto2D p4; p4.moverOPonto(16,24);
    Ponto2D *aux = new Ponto2D(p4);
    pilhaDePontos.inserir(aux);

    pilhaDePontos.inserir(p1);
    pilhaDePontos.inserir(p2);pilhaDePontos.inserir(p3);

    Ponto2D pontoOut;

    while( pilhaDePontos.quantidadeDeElementos() > 0)
    { pontoOut = pilhaDePontos.retirar();
      std::cout << pontoOut << std::endl;
    }
      std::cout << "fim";
     return 0;
}
```

Quadro 6 - Arquivo auxiliar ao exemplo: Ponto2D.h

```
/*
 * Ponto2D.h
 *
 *  Created on: Apr 19, 2014
 *      Author: robinsonnoronha
 */

#ifndef PONTO2D_H_
#define PONTO2D_H_
#include <ostream>

class Ponto2D {
private:
         int coordenada_X;
         int coordenada_Y;
public:
      Ponto2D();
      virtual ~Ponto2D();
      void moverOPonto(int,int);
      double calcularADistancia(Ponto2D *);
      friend std::ostream & operator <<( std::ostream &, Ponto2D &);
      bool operator ==(Ponto2D &);
};
#endif /* PONTO2D_H_ */
```

Quadro 7 - Arquivo auxiliar ao exemplo: Ponto2D.h

```
#include "Ponto2D.h"
#include <cmath>

Ponto2D::Ponto2D() {
    this->coordenada_X = 0;
    this->coordenada_Y = 0;
}

Ponto2D::~Ponto2D() { }

void Ponto2D::moverOPonto(int nx, int ny)
{ this->coordenada_X = nx; this->coordenada_Y = ny;  }

double Ponto2D::calcularADistancia(Ponto2D *outro)
{
      int DeltaX = this->coordenada_X - outro->coordenada_X;
      int DeltaY = this->coordenada_Y - outro->coordenada_Y;

    return sqrt( (double) (DeltaX*DeltaX + DeltaY*DeltaY));
}

bool Ponto2D::operator ==(Ponto2D &v)
{
    bool saida = false;

    if ( (v.coordenada_X == this->coordenada_X)
      && (v.coordenada_Y == this->coordenada_Y) )
                        saida = true;
      return saida;
}

std::ostream & operator <<( std::ostream &out, Ponto2D &v)
{
      out << "(" << v.coordenada_X << ", " << v.coordenada_Y << ")";
      return (out);
}
```

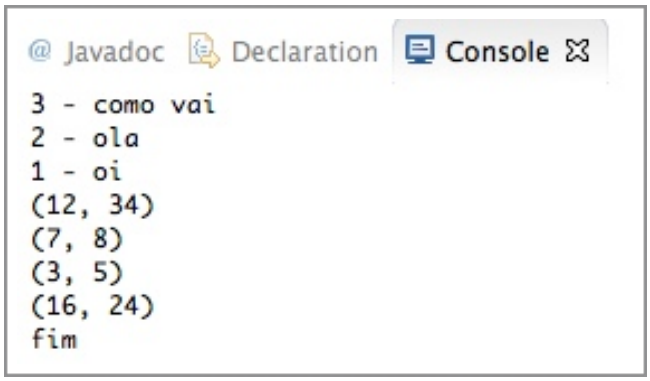

Figura 4 – Resultado da Execução dos Códigos C++ apresentados nos Quadros 4, 5 , 6 e 7.

4.2 Implementação em Java

Quadro 8 - Exemplo de Template que implementa a Pilha Genérica ilustrada na Figura 2 - Código Java

```
class Pilha <T> {

    private class Node
    {  public T valor;
       public Node next;
       Node( T v){ valor = v; next = null;}
    };

    Node topo;

    Pilha(){ topo=null;}

    Pilha(T v)  { topo = null; inserir(v); }

    public void inserir(T dado)
    {
      Node novo = new Node(dado);
      novo.next = topo;
      topo = novo;
    }

    public T retirar()
    {
      T saida = topo.valor;
      topo = topo.next;
      return saida;
    }

    public int quantidadeDeElementos()
    {
      int saida = 0;
      Node andarilho = topo;
      while (andarilho != null)
      {
            andarilho = andarilho.next;
            saida++;
      }
      return saida;
    }
}
```

Universidade Tecnológica Federal do Paraná.
Departamento Acadêmico de Eletrônica
prof. Robinson Vida Noronha (vida@utfpr.edu.br) e Bruno Chang

Quadro 9 - Exemplo de Uso do Template Pilha - Java

```
class NumeroComplexo
{
      int real, imaginario;
      NumeroComplexo(int r, int i) {real = r; imaginario = i;}
      public String toString(){ return "(" + real + ") + j(" + imaginario
                                  + ")"; }
}
public class Main {

public static void main(String[] args) {

 String nome[] = new String[5];
 nome[0]= "Ana";  nome[1]= "Pedro"; nome[2]= "Carlos";
 nome[3]= "Egberto";  nome[4] = "Maria";

 Pilha <String> amigos = new Pilha();

 for (int i = 0; i < nome.length; i++)  amigos.inserir(nome[i]);

  System.out.println("Nomes inseridos na Pilha.” +
  “ Quantidade de elementos na pilha=" + amigos.quantidadeDeElementos() );

  for (int i = 0 ; i < nome.length; i++)
      System.out.println("> retirei " + amigos.retirar()
       + ". Resultado, pilha com " + amigos.quantidadeDeElementos()
       + " elementos armazenados.");

/*  segundo exemplo de uso: Pilha de Numeros Complexos  */
 System.out.println("\n\n Segundo Exemplo da Pilha: Numeros Complexos.");

// definicao da pilha de numeros complexos
 Pilha <NumeroComplexo>  numeros = new Pilha();

// criacao dos elementos que serao guardados na pilha
// definicao de array de objetos
  NumeroComplexo aux[] = new NumeroComplexo[3];

// instancias dos elementos do array
  aux[0] = new NumeroComplexo(4,5);  aux[1] = new NumeroComplexo(7,8);
  aux[2] = new NumeroComplexo(9,13);

  for (int i = 0; i < aux.length; i++)  numeros.inserir(aux[i]);

  System.out.println("Numeros Complexos inseridos na Pilha.”
    + “ Quantidade de elementos na pilha=" + amigos.quantidadeDeElementos());

  for (int i = 0 ; i < aux.length; i++)
      System.out.println("> retirei " + numeros.retirar()
           + ". Resultado, pilha com " + numeros.quantidadeDeElementos()
           + " elementos armazenados.");

 }
}
```

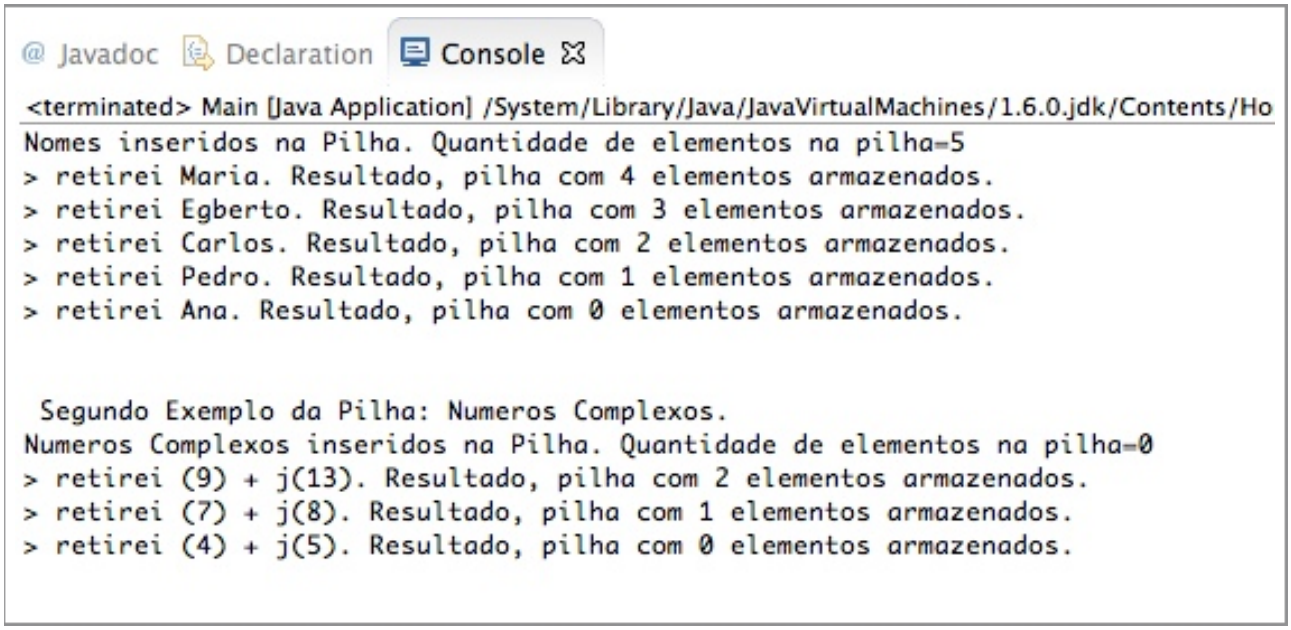

Figura 5 – Resultado da Execução dos Códigos Java apresentados nos Quadros 8 e 9.

Referências:

[1] Budd, T. (2002). An introduction to Object-Oriented Programming . Third Edition. Addison Wesley .

[2] Site: http://docs.oracle.com/javase/tutorial/java/generics/types.html consultado em 19 de abril de 2014.