**ETEC - Jardim Ângela**

**Nome do Aluno: Vitor Francisco Lamounier**

**2º Modulo – Técnico de Informática**

Disciplina de TOO

Professora: Edina

2015

Vitor Francisco Lamounier

Técnicas de Orientação de Objetos:

Tipos de Programação

São Paulo

2015

SUMÁRIO

[Programação Estruturada 3](#_Toc412035781)

[Exemplo 1 - Repetição 4](#_Toc412035782)

[Exemplo 2 - Decisão 5](#_Toc412035783)

[Programação Orientada a Objetos 6](#_Toc412035784)

[Exemplo 1 – Encapsulamento Java 7](#_Toc412035785)

[Exemplo 2 – Encapsulamento C++ 8](#_Toc412035786)

[Programação Modular 9](#_Toc412035787)

[Exemplo 1 – Função 10](#_Toc412035788)

[Exemplo 2 – Sub Rotina 11](#_Toc412035789)

[Programação Recursiva 12](#_Toc412035790)

[Exemplo 1 – Algoritmo Recursivo 13](#_Toc412035791)

[Exemplo 2 – Fatorial Recursivo 14](#_Toc412035792)

[Conclusão 15](#_Toc412035793)

[Referências 16](#_Toc412035794)

# Programação Estruturada

Programação estruturada é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e iteração (esta última também é chamada de repetição), desenvolvida por Michael A. Jackson no livro "Principles of Program Design" de 1975.

Tendo, na prática, sido transformada na programação modular, a programação estruturada orienta os programadores para a criação de estruturas simples nos programas, usando as sub-rotinas e as funções. Foi a forma dominante na criação de software anterior à programação orientada por objetos.

Apesar de ter sido sucedida pela programação orientada por objetos, pode-se dizer que a programação estruturada ainda é muito influente, uma vez que grande parte das pessoas ainda aprendem programação através dela. Para a resolução de problemas relativamente mais simples e diretos, a programação estruturada é muito eficiente. Além disso, por exigir formas de pensar relativamente complexas, a programação orientada a objetos até hoje ainda não é bem compreendida ou usada pela maioria.

Exemplos:

* Escolha – Cobol
* Decisão – PHP

## Exemplo 1 - Repetição

Um algoritmo que permite realizar qualquer uma das quatro operações pela estrutura escolha.

Código:

identification division.

program-id. LacoRepeticao.

author. Matheus\_Vinicius .

data division.

working-storage section.

77 numero pic 999 value zero.

77 pares pic 999 value zero.

77 impares pic 999 value zero.

77 d pic 999 value zero.

77 resto pic 999 value zero.

77 pp pic 99v99 value zero.

77 pi pic 99v99 value zero.

77 i pic 99 value zero.

77 ms pic ZZ9.99 value zero.

procedure division.

inicio.

display erase

perform varying i from 1 by 1 until i>10

display "numero" i

accept numero

compute d=numero/2

compute resto=numero - d\*2

if resto <> 0 then

add 1 to impares

else

add 1 to pares

end-if

end-perform

compute pp=((pares /10) \* 100)

compute pi=((impares / 10) \* 100)

display erase

display "Qtd de numeros pares :" at 1010 pares

display "Qtd de numeros impare :" at 1110 impares

move pp to ms

display "Percentual de numeros pares: " at 1210 ms

move pi to ms

display "Percentual de numeros impares: " at 1310 ms

stop run

## Exemplo 2 - Decisão

PHP usando uma estrutura de decisão para exibir uma mensagem conforme a hora do dia.

Código:

if($hora <= 12)

echo “Está de manhã”;

else if($hora > 12 && $hora <= 18)

echo “Está de tarde”;

else

echo “Está de noite”;

# Programação Orientada a Objetos

A orientação a objetos é um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Em alguns contextos, prefere-se usar modelagem orientada ao objeto, em vez de programação. De fato, o paradigma "orientação a objeto", tem bases conceituais e origem no campo de estudo da cognição, que influenciou a área de inteligência artificial e da linguística, no campo da abstração de conceitos do mundo real. Na qualidade de método de modelagem, é tida como a melhor estratégia para se eliminar o "gap semântico", dificuldade recorrente no processo de modelar o mundo real do domínio do problema em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio. Facilitaria a comunicação do profissional modelador e do usuário da área alvo, na medida em que a correlação da simbologia e conceitos abstratos do mundo real e da ferramenta de modelagem (conceitos, terminologia, símbolos, grafismo e estratégias) fosse a mais óbvia, natural e exata possível.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

Exemplos:

* Encapsulamento – Java
* Encapsulamento – C++

## Exemplo 1 – Encapsulamento Java

Encapsulamento de um objeto em Java.

Código:

private int id;

public int GetId()

{

return id;

{

public void SetId(int id)

{

this.id = id;

}

## Exemplo 2 – Encapsulamento C++

Encapsulamento com linguagem C++.

Código:

private:

int id;

public:

int GetId() const;

void SetId(int const id);

# Programação Modular

Modularização em tecnologia da informação é um conceito onde o sistema ou software é divido em partes distintas. Compõe o ferramental necessário para um programa mais legível com uma melhor manutenção e melhor desempenho por meio da programação estruturada.

Pode ser caracterizado da seguinte forma: Elemento separadamente endereçável do sistema, menor parte do sistema que realiza uma função completa independente de outras funções, conjunto de instruções de um programa que pode ser chamado por um nome, sendo ideal que para os outros módulos seja uma caixa preta.

A ideia de dividir os programas em módulos surgiu no final da década de 1960. Os países desenvolvidos estavam então passando por uma “crise de software”. Essa crise se deu ao rápido desenvolvimento do hardware ao passo que as técnicas aplicadas ao desenvolvimento de software avançavam lentamente. O Desenvolvimento de sistemas sem uma metodologia para a construção de programas, geralmente resulta em um software com vários erros e com alto custo de desenvolvimento que, consequentemente exige um custo elevado para sua correção e manutenção futuras. A modularização de programas juntamente com outras técnicas de programação integram o ferramental para a elaboração de programas visando, principalmente, os aspectos de confiabilidade, legibilidade, manutenção e flexibilidade.

Um modulo pode ser definido como um grupo de comandos, constituindo um trecho de algoritmo, com uma função bem definida e o mais independente possível em relação ao resto do algoritmo. A maneira mais intuitiva de proceder à modularização é definir um modulo principal de controle e modulo especifico para as funções do algoritmo. O ideal é que os módulos não sejam grandes demais, pois senão acabam sendo multifuncionais e de difícil compreensão, de modo que o modulo deve ser implementado apenas as estruturas de dados necessários para atingir ao objetivo do modulo.

Exemplos:

* Função - C
* Sub Rotina - C

## Exemplo 1 – Função

Função de espera para um sistema feito em linguagem C.

Código:

#include <stdlib.h>

#include <conio.h>

#include <dos.h>

#include <stdio.h>

void EsperaEnter() // Definição da função "EsperaEnter"

{

int tecla;

printf("Pressione ENTER\n");

do{

tecla = getch();

if (tecla !=13) // Se nao for ENTER{

sound(700); // Ativa a emissão de um BEEP

delay(10); // Mantém a emissão do som por 10 ms

nosound(); // Para de emitir o som

}

} while(tecla != 13); // 13 e' o codigo ASCII do ENTER

}

void main()

{

EsperaEnter(); // Chamada da função definida antes

}

## Exemplo 2 – Sub Rotina

Uma sub-rotina sonora em linguagem C.

Código:

void Tocarsom(){

int contador;

for(contador = 100; contador < 1000; contador += 50){

sound(contador);

delay(100);

}

nosound();

}

# Programação Recursiva

Em ciência da computação, a recursividade é a definição de uma sub-rotina (função ou método) que pode invocar a si mesma. Um exemplo de aplicação da recursividade pode ser encontrado nos analisadores sintáticos recursivos para linguagens de programação. A grande vantagem da recursão está na possibilidade de usar um programa de computador finito para definir, analisar ou produzir um estoque potencialmente infinito de sentenças, designs ou outros dados.

Um método comum de simplificação consiste em dividir um problema em subproblemas do mesmo tipo. Como técnica de programação, isto se denomina divisão e conquista, e constitui a chave para o desenvolvimento de muitos algoritmos importantes, bem como um elemento fundamental do paradigma de programação dinâmica.

Praticamente todas as linguagens de programação usadas hoje em dia permitem a especificação direta de funções e procedimentos recursivos. Quando uma função é invocada, o computador (na maioria das linguagens sobre a maior parte das arquiteturas baseadas em pilhas) ou a implementação da linguagem registra as várias instâncias de uma função (em muitas arquiteturas, usa-se uma pilha de chamada, embora outros métodos possam ser usados). Reciprocamente, toda função recursiva pode ser transformada em uma função iterativa usando uma pilha.

Toda função que puder ser produzida por um computador pode ser escrita como função recursiva sem o uso de iteração; reciprocamente, qualquer função recursiva pode ser descrita através de iterações sucessivas.

Algumas linguagens desenvolvidas para programação lógica e programação funcional permitem recursões como única estrutura de repetição, ou seja, não podem usar laços tais como os produzidos por comandos como for, while ou repeat. Tais linguagens geralmente fazem uma recursão em cauda tão eficiente quanto a iteração, deixando os programadores exprimirem outras estruturas de repetição (tais como o map e o for do Scheme) em termos de recursão.

Exemplo

* Algoritmo Recursivo – Java
* Fatorial Recursivo – Ruby

## Exemplo 1 – Algoritmo Recursivo

O algoritmo de Fibonacci em Java para usar a Recursividade.

Código:

import java.util.Scanner;

import java.util.regex.\*;

public class FibonacciRecursivo {

public static void main (String[] args) {

solicitarLimite();

}

public static void solicitarLimite() {

System.out.println("Digite o limite para a sequencia: ");

Scanner limite = new Scanner(System.in);

String ate = limite.nextLine();

Pattern padrao = Pattern.compile("^([0-9])+$");

Matcher ok = padrao.matcher(ate);

if (ok.matches()) {

int limiteSeq = Integer.parseInt(ate);

int n1 = 0;

int n2 = 1;

calcularFiboRecursivo(n1, n2, limiteSeq);

} else {

System.out.println("Digite um número inteiro válido.");

}

}

public static void calcularFiboRecursivo(int n1, int n2, int limiteSeq) {

int daVez = n1 + n2;

System.out.println(n2);

n1 = n2;

n2 = daVez;

if (daVez <= limiteSeq) {

calcularFiboRecursivo(n1, n2, limiteSeq);

}

}

}

## Exemplo 2 – Fatorial Recursivo

Algoritmo em Ruby que calcula o Fatorial de um número pelo método da Recursividade

Código:

def fatorial(n)

if n <= 0

return 1

else

return n \* fatorial(n - 1)

end

end

def mostrar\_resultado(n)

puts "Resultado: #{fatorial(n)}"

end

def inicia\_programa

puts "Digite um número para calcular o fatorial"

n = gets.chomp

if n =~ /^[0-9]+$/

mostrar\_resultado(n.to\_i)

else

puts "Número inválido!"

puts "Digite um número inteiro positivo:"

puts "----------------------------------"

inicia\_programa

end

end

# Conclusão

A programação estruturada foi desenvolvida por Michael Anthony Jackson no livro “Principles of Program Design” e prova que qualquer problema pode ser resolvido por um computador usando apenas três estruturas: sequência, decisão (escolha) e iteração (repetição).

A programação orientada a objetos cria métodos (comportamentos) e atributos (estados) para classes que serão usadas em um sistema, incluindo o relacionamento que podem ter com outros objetos.

A programação modular separa elementos que exercem funções independentes para que não interfiram no desenvolvimento do sistema, porém os elementos ainda devem estar preparados para funcionar quando suas funções forem exigidas.

A programação recursiva monta algoritmos que são ativados automaticamente após detectarem podem realizar alguma tarefa.

# Referências

VINICIUS, Matheus. *Lógica de Software.* Em: <http://logicadesoftware.blogspot.com.br/2012/ 02/laco-de-repeticao-cobol-identifica.html> . Acesso em: 18 fevereiro 2015.

GASPAROTTO, Henrique Machado. *Os 4 pilares da Programação Orientada a Objetos* Em: <http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 18 fevereiro 2015.

BERTOL, Omero Francisco. *Introdução à Programação Estruturada.* Em: <http:// www.devmedia.com.br/introducao-a-programacao-estruturada/24951>. Acesso em: 18 fevereiro 2015.

*Programação Estruturada.* Em: <http://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o\_ estruturada>. Acesso em: 18 de fevereiro 2015

*Modularidade.* Em: http://pt.wikipedia.org/wiki/Modularidade. Acesso em: 18 fevereiro 2015.

*Recursividade (ciências da computação).* Em: <http://pt.wikipedia.org/wiki/Recursividade\_ %28ci%C3%AAncia\_da\_computa%C3%A7%C3%A3o%29>. Acesso em: 18 fevereiro 2015.

*Orientação a objetos.* Em: <http://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o\_a\_objetos>. Acesso em: 18 fevereiro 2015.