GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

ESCOLA ESTADUAL DE EDUCAÇÃO PROFISSIONAL - EEEP

ENSINO MÉDIO INTEGRADO À EDUCAÇÃO PROFISSIONAL

CURSO DE INFORMÁTICA

LÓGICA DE PROGRAMAÇÃO

GOVERNO DO
ESTADO DO CEARÁ
Secretaria da Educação

Governador
Cid Ferreira Gomes

Vice Governador
Domingos Gomes de Aguiar Filho

Secretária da Educação
Maria Izolda Cela de Arruda Coelho

Secretário Adjunto
Maurício Holanda Maia

Secretário Executivo
Antônio Idilvan de Lima Alencar

Assessora Institucional do Gabinete da Seduc
Cristiane Carvalho Holanda

Coordenadora da Educação Profissional – SEDUC
Andréa Araújo Rocha

GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

Coordenação Técnica Pedagógica
Renanh Gonçalves de Araújo

Equipe de Elaboração
Adriano Gomes da Silva
Cíntia Reis de Oliveira
Fernanda Vieira Ribeiro
Francisco Aislan da Silva Freitas
João Paulo de Oliveira Lima
Liane Coe Girão Cartaxo
Mirna Geyla Lopes Brandão
Moribe Gomes de Alcântara
Niltemberg Oliveira Carvalho
Paulo Ricardo do Nascimento Lima
Renanh Gonçalves de Araújo
Renato William Rodrigues de Souza

Colaboradores
Maria Analice de Araújo Albuquerque
Maria Danielle Araújo Mota
Sara Maria Rodrigues Ferreira Feitosa

Sumário

Caro Aluno,
Neste manual são apresentados os conceitos introdutórios sobre a tarefa de programar computadores, a necessidade do uso da lógica na programação, a importância de se abstrair e modelar os problemas antes de partir para as soluções. Por fim, são apresentadas algumas dicas úteis que podem ser utilizadas na solução de problemas em geral.

INTRODUÇÃO

Um programa de computador é um produto resultante da atividade intelectual de um programador. Essa atividade, por sua vez, depende de um treinamento prévio em abstração e modelagem de problemas, bem como o uso da lógica na verificação das soluções.

Programar é como escrever um texto para que se possa escrever corretamente você primeiramente pensa e analisa o vocabulário depois inicia o procedimento de escrever colocando cada palavra no seu devido lugar e usando a sintaxe correta, no mundo da programação é de extrema importância à lógica, pois através dela adquirimos a capacidade de escrever programas em qualquer linguagem de programação, é isso mesmo o que muda de uma linguagem para outra é a sintaxe.

Recursos que iremos utilizar, adotaremos o software Portugol IDE 2.3 para trabalhar com português estruturado e a linguagem de programação Java usando a ide do Netbeans 6.9.1. Esta disciplina é a primeira etapa para que o aluno possa adquirir as competências relacionadas ao desenvolvimento de softwares, todos os algoritmos utilizados serão apresentados em português estruturado e em Java.

Fase I - Introdução a Lógica de Programação

Aula 1 - Introdução a Lógica de Programação

À LÓGICA DE PROGRAMAÇÃO

1.1 - CONCEITOS BÁSICOS

Nesta disciplina, iniciaremos nossos estudos sobre Lógica de Programação. Mas, antes de começarmos, seria útil uma reflexão sobre o significado da palavra “Lógica”. Assim, o que é Lógica?

Lógica trata da correção do pensamento. Como filosofia, ela procura saber por que pensamos assim e não de outro jeito. Com arte ou técnica, ela nos ensina a usar corretamente as leis do pensamento.
O filósofo grego Aristóteles é considerado o criador da lógica, em sua época denominava-se ***razão,*** depois que a palavra ***lógica*** começou a ser utilizada, esta tem origem do grego ***logos*** que significa linguagem racional.

Poderíamos dizer também que a Lógica é a arte de pensar corretamente e, visto que a forma mais complexa do pensamento é o raciocínio, a Lógica estuda ou tem em vista a correção do raciocínio.
Podemos ainda dizer que a lógica tem em vista a ordem da razão. Isto dá a entender que a nossa razão pode funcionar desordenadamente, pode pôr as coisas de pernas para o ar. Por isso a Lógica ensina a colocar Ordem no Pensamento.

Desordem: “E se eu escolher aquilo?” ou “É, mais eu não tinha pensado nisso”.
Enfim, lógica é ciência que coloca a cabeça para funcionar corretamente.
Para chegarmos à conclusão de algo utilizamos as premissas que são conhecimentos prévios, desta forma organizamos o pensamento, com a organização do mesmo é que concluímos se algo é verdadeiro ou falso.

Utilizamos a lógica de forma natural em nosso dia-a-dia. Por exemplo:

a) Sei que o livro está no armário.
Sei que o armário está fechado
Logo, concluo que tenho de abrir o armário para pegar o livro.
Vamos observar neste exemplo as premissas e os pontos os quais levam a conclusão deste fato.

1ª. (premissa) Sei que o **A** está no **B**.
2ª. (premissa) Sei que o **B** está fechado.
3ª. (conclusão) Logo, concluo que tenho de abrir o armário para pegar o livro.

Sendo A o livro ou qualquer outra coisa que tenho que pegar em B(armário), tenho o conhecimento prévio de que o que quero pegar está no armário e o mesmo encontra-se fechado.
Neste exemplo do dia-a-dia tenho duas premissas que através delas chego a uma conclusão.

b) Sei que sou mais velho que João. (premissa)
Sei que João é mais velho que José. (premissa)
Então, concluo que eu sou mais velho que José. (conclusão)
Neste exemplo só consigo chegar a uma conclusão de que sou mais velho que alguém se existirem as duas premissas, só com apenas uma não conseguiria chegar a uma conclusão.

1. Sejam os seguintes fatos:
• Todos os filhos de José são mais altos do que Maria.
• Antônio é filho de José.

Então, o que podemos concluir logicamente?

Considere os fatos abaixo:

• Pedro é aluno da EEEP.
• Para ser aprovado, um aluno da EEEP precisa obter nota maior ou igual a 6,0 e comparecer a mais de 75% das aulas.
• Pedro compareceu a todas as aulas e obteve nota igual a 8,0.

Então, o que podemos concluir?

Exercício Prático

1 - Através das premissas a seguir, assinale as sentenças que correspondem à conclusão correta.

Se o semáforo com a luz vermelha é para o motorista parar e o verde para seguir, estando eu a pé para atravessar a rua então concluo que:

a) Posso atravessar a rua com a luz vermelha.

b) O semáforo tem duas luzes.

c) Só devo atravessar a rua com a luz verde.

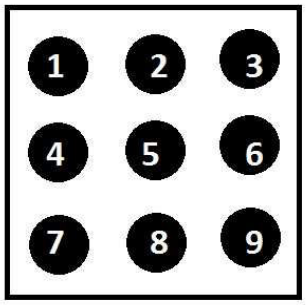
2 - Patos são animais. Patos têm duas patas. Logo:

a) Todo o animal tem duas patas.

b) Patos têm duas patas.

c) Patos tem bico.

3 - Desafio dos nove pontos o objetivo é traçar quatro linhas retas passando por todos os nove pontos, sem tirar o lápis/caneta do papel. Para facilitar o raciocínio e a resolução, marque os nove pontos em uma folha de papel e tente resolver.



Aula 2 – Introdução aos Algoritmos

2.1 - O que é um ALGORITMO?

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por seqüências lógicas. Por exemplo:

“Chupar uma bala”.

1. Pegar a bala.
2. Retirar o papel.
3. Chupar a bala.
4. Jogar o papel no lixo.

Exercício Prático

1 - Crie uma sequência lógica para tomar banho:

2 - Descreva com detalhes a sequência lógica para Trocar um pneu de um carro.

3 - Faça um algoritmo para trocar uma lâmpada. Descreva com detalhes:

2.2 – TIPOS DE ALGORITMOS

Os tipos de algoritmos mais usados são descrição narrativa, fluxograma e Pseudocódigo ou Portugol, agora vamos conhecer um pouco destes tipos e nos próximos tópicos nos aprofundarmos.

Descrição narrativa

Utiliza linguagem natural;

Vantagem: Não é necessário aprender nenhum conceito novo, é como estivéssemos falando ou escrevendo os detalhes de algo para outra pessoa.

Desvantagem: Permite várias interpretações, dificultando transcrição para programa.

Descrição narrativa (Exemplo)

Ler dois números e calcular a média

1 Ler os dois números.
2 Calcular a média.
3 Mostrar o resultado da média.

Fluxograma: Utiliza elementos gráficos, que nos próximos tópicos abordaremos com mais detalhes este assunto.

Vantagem: Entendimento de gráficos é mais fácil que de textos.

Desvantagem: Necessário aprender simbologia e não apresenta detalhes para transcrever para programa.

Pseudocódigo ou Portugol.

Utiliza uma linguagem com regras definidas com uma estrutura formal também conhecido como português estruturado, na aula 2.2 será discutido.

Vantagem: Transcrição para programa (linguagem de computador) é praticamente imediata.

Desvantagem: Necessário aprender regras.

2.3 PSEUDOCÓDIGO

Os algoritmos são descritos em uma linguagem chamada **pseudocódigo**. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo Java, estaremos gerando código em Java. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.
Utilizaremos o software Portugol Ide para desenvolver os nossos algoritmos em pseudocódigo, que pode ser feito o download no site http://www.dei.estt.ipt.pt/portugol.

REGRAS PARA CONSTRUÇÃO DO ALGORITMO

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva.

Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase.
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática.
- Usar frases curtas e simples.
- Ser objetivo.
- Procurar usar palavras que não tenham sentido dúbio.

FASES

No capítulo anterior vimos que **ALGORITMO** é uma sequência lógica de instruções que podem ser executadas.

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

COMO FAZER ARROZ DOCE
ou então
CALCULAR O SALDO FINANCEIRO DE UM ESTOQUE

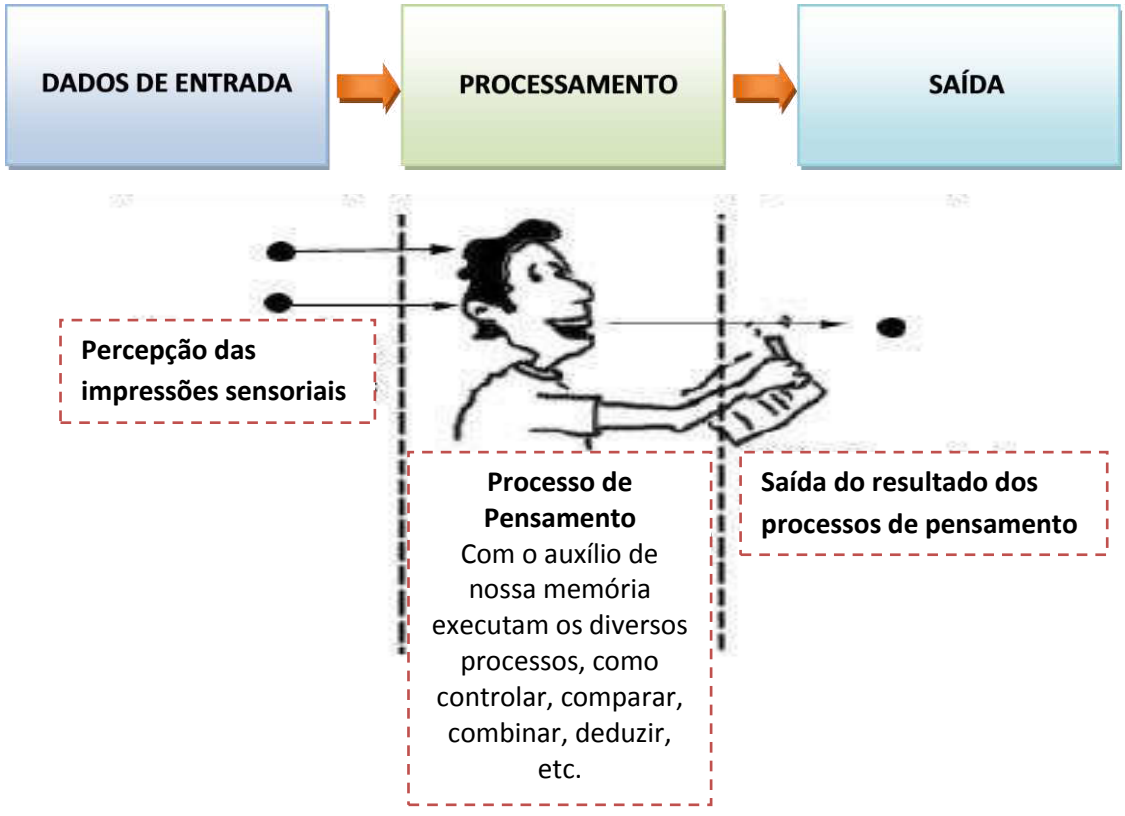
Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais.

Entrada: São os dados necessários para a resolução do problema proposto;

Processamento: São os processamentos utilizados para chegar ao resultado final;

Saída: São os dados processados apresentando o resultado para o problema proposto;

Analogia com o homem



EXEMPLO DE ALGORITMO

Imagine o seguinte problema: Calcular a média final dos alunos da 3ª Série. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$\text{Média Final} = \frac{P1+P2+P3+P4}{4}$$

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada?
R: Os dados de entrada são P1, P2, P3 e P4

b) Qual será o processamento a ser utilizado?
R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

$$\frac{P1+P2+P3+P4}{4}$$

c) Quais serão os dados de saída?
R: O dado de saída será a média final

ALGORITMO (SEQUÊNCIA DE EXECUÇÃO)

Receba a nota da prova1.
Receba a nota de prova2.
Receba a nota de prova3.
Receba a nota da prova4.
Some todas as notas e divida o resultado por 4.
Mostre o resultado da divisão.

Em pseudocódigo no portugol o algoritmo ficaria desta forma.

PortugolIDE

```
inicio
variavel real prova1
variavel real prova2
variavel real prova3
variavel real prova4
variavel real media
ler prova1
ler prova2
ler prova3
ler prova4
media<- (prova1+prova2+prova3+prova4)/4
escrever "sua media é "
escrever media

fim
```

Estrutura de um algoritmo independente do problema os algoritmos tem a mesma estrutura.

calcular_media	**Identificação do algoritmo**

variavel real prova1 **variavel** real prova2 **variavel** real prova3 **variavel** real prova4 **variavel** real media	**Declaração de variáveis**

ler prova1 **ler** prova2 **ler** prova3 **ler** prova4 **media**<- (prova1+ prova2+ prova3+ prova4)/4 **escrever** "sua media é " **escrever** media	**Corpo do algoritmo**

Identificação do algoritmo: Todo algoritmo ele deve ser identificado, abaixo algumas regras básicas;

- ✓ Não utilizar espaços entre as letras ou caracteres especiais como acentos, símbolos (@#%&*?:/) entre outros.
- ✓ Para identificar um algoritmo com duas palavras, por exemplo “calcular media“ usar o underline o correto ficaria calcular_media.
- ✓ Não utilizar palavras reservadas como as que são utilizadas para representar ações especificas como **ler**, **variavel**, **escrever** no portugol ide as palavras reservadas são destacadas em negrito.
- ✓ Não utilizar números no inicio da identificação do algoritmo como, por exemplo; “1exemplo” o correto seria “exemplo1”.
- ✓ Usar nomes coerentes para identificação de algoritmos, nomes os quais possam identificar o que o algoritmo vai fazer.

Declaração de variáveis: As variáveis que serão utilizadas na resolução de problemas, devem ser declaradas, que são as informações relacionadas à resolução do problema, com relação a variáveis iremos ver em detalhes na aula 6.

Corpo do algoritmo: No corpo do algoritmo deve ser escrito todos os passos para a resolução de problemas, como por exemplo;

- ✓ Entrada de valores para as variáveis.
- ✓ Operações de atribuição tais como lógicas e aritméticas.
- ✓ Laços de repetição.
- ✓ Exibição de resultados.

Mais o que são variáveis?

Variáveis são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado.

Estas devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e às vezes um valor inicial.

Tipos podem ser, por exemplo: inteiros, reais, caracteres, etc. As expressões combinam variáveis para calcular novos valores, na aula 6 iremos ver mais detalhes e a aplicação das mesmas em uma linguagem de programação.

Tipos de dados básicos usados em pseudocódigo.

Abaixo temos uma tabela com os principais tipos de dados que iremos utilizar em pseudocódigo usando o software Portugol ide, na aula 6 iremos ver os tipos primitivos e suas representações em uma linguagem de programação.

No Portugol ide usamos a palavra **variavel** para declarar uma variável como no exemplo anterior (**variavel real** prova1) na tabela podemos ver os tipos de dados utilizados.

Tipo	Descrição	Valores	Valor por defeito
Inteiro	Valores ordinais definidos com quatro bits	-2 147 483 648 2 147 483 647	0
Real	Valores com parte decimal definidos com 64 bits	-1.7 E 308 1.7 E 308	0.0
Lógico	Valore lógicos - 1 bit	verdadeiro falso	falso
Carácter	Caracteres da Tabela ASCII	ASCII(0) ASCII(255)	" " (espaço)
Texto	Conjuntos de caracteres	"Sequências de caracteres" "entre aspas"	"" (vazio)

Atribuição de valores.

No portugol ide utilizamos este símbolo <- para atribuir valor a uma variável, no exemplo anterior usamos este para dizer que a variável media irá receber os valores da soma e divisão de prova1, prova2, prova3, prova4.

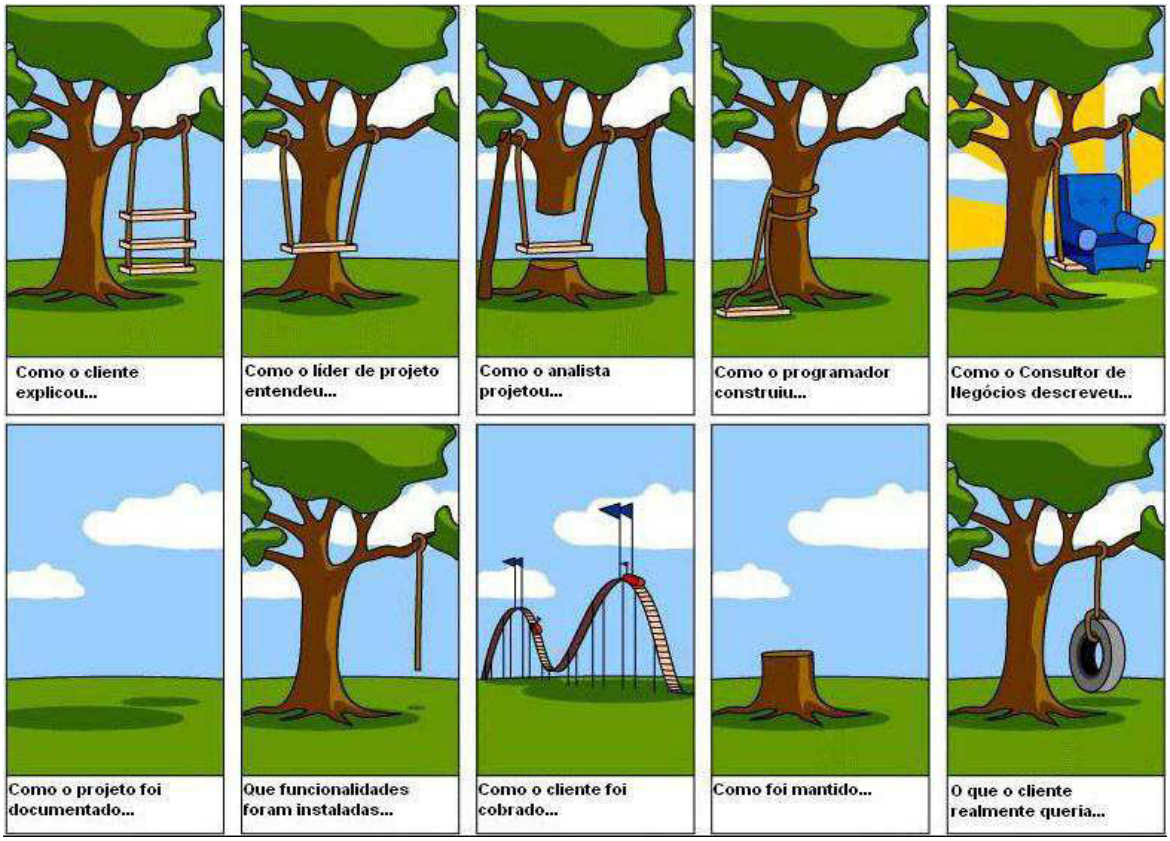
Escrevendo e lendo dados.

Para escrever algo utilizamos a palavra **escrever,** se quisermos escrever algum texto digitamos a palavra **escrever** (“mensagem a ser exibida”), se for uma variável então digitamos **escrever** e o nome da variável declarada.
Para ler uma variável que recebeu uma entrada do usuário digitamos a palavra **ler** acompanhada do nome da variável declarada.

Para construção de um algoritmo são necessários os passos descritos a seguir:

1. Ler atentamente o enunciado, destacando os pontos mais importantes;
2. Definir os dados de entrada, ou seja, quais dados serão fornecidos;
3. Definir o processamento, ou seja, quais cálculos serão efetuados e quais as restrições para esses cálculos. O processamento é responsável pela transformação dos dados de entrada em dados de saídas;
4. Definir os dados de saídas, ou seja, quais dados serão gerados depois do processamento;
5. Construir o algoritmo utilizando um dos tipos descritos;
6. Testar o algoritmo utilizando as informações.

Importância da Análise de Processos



Exercício Prático

Praticando no Portugol ide, vamos praticar! Desenvolvendo os algoritmos abaixo no Portugol ide.

1 - Escreva um algoritmo que receba 2 números e exiba o resultado da sua soma.

2 - Escreva um algoritmo que receba 2 números e ao final exiba o resultado da subtração, multiplicação e divisão dos números lidos.Escreva um algoritmo que receba o ano atual, ano de nascimento de uma pessoa e mostre a sua idade.

3 - O Sr. João necessita saber o consumo médio de um automóvel, e solicitou para você desenvolver um algoritmo que sendo fornecido a distancia total percorrida pelo automóvel e o total de combustível gasto, mostrar o consumo do automóvel.

4 - Escreva um algoritmo que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o seu nome, o salário fixo e o salário no final do mês.

5 - Ler uma temperatura em graus Celsius e apresenta-la em Fahrenheit. A fórmula de conversão é: F=(9*C+160)/5, sendo F Fahrenheit e C Celsius.

6 - A loja ELETROMOVEIS esta vendendo os seus produtos no cartão em 5 vezes sem juros, Faça um algoritmo que receba o valor da compra e mostre o valor das parcelas.

2.4 - Teste de Mesa

Para testar se as funcionalidades implementadas em um algoritmo estão corretas é necessário testar o algoritmo, verificando o conteúdo das variáveis passo a passo. Para efetuar esta tarefa, costuma-se utilizar o chamado Teste de Mesa ou Teste Exaustivo. Realizar este teste significa seguir passo a passo as instruções do algoritmo, de maneira precisa, para verificar se o que foi implementado está correto ou não, a partir dos resultados gerados e dos valores parciais. Este teste permitirá que o programador visualize o comportamento de todo o processo, cheque se o algoritmo está correto e corrija eventuais erros, se existirem. Em Informática, dá se o nome de “bugs” aos erros de um programa. O processo de identificação e correção dos erros denomina-se “debugging”. Os erros podem ser dos seguintes tipos:

Erros Sintáticos - ocorrem quando as instruções do programa não são escritas de acordo com a sintaxe da linguagem sendo usada. Por exemplo: se eu esquecesse um **fim se** de um comando **se**, o comando estaria incompleto. Não estaria definido conforme a sintaxe da linguagem. Por isso, ocorreria o erro sintático. Este tipo de erro impede a execução do algoritmo ou programa.

Erros Semânticos - (ou lógicos) - ocorrem quando o código escrito pelo programador não gera o resultado desejado por este. Ou seja, o código está sintaticamente correto, mas o resultado gerado pelo algoritmo não está correto. Erros de lógica podem ser simplesmente uso incorreto de operadores (+ no lugar de -, usar o operador OU no lugar de usar o E), atribuições erradas (por exemplo, pedir para digitar o nome da pessoa e guardar o que for digitado na variável endereço), etc.

Os erros sintáticos são fáceis de encontrar e corrigir porque, geralmente, o compilador se encarrega de apontá-los e, normalmente, dá uma indicação do tipo de erro. O programa só é executado quando não existem mais erros sintáticos. Já os erros semânticos são de detecção mais difícil, uma vez que os compiladores não podem encontrar erros de lógica, já que não tem conhecimento sobre o que o programador deseja fazer. Dessa forma, erros de lógica só podem ser encontrados e remediados pelo programador. Esse tipo de erro pode fazer com que o programa exiba comportamentos inesperados.

E como se faz o teste de mesa?

1. Leia o algoritmo que foi escrito.

2. Crie uma coluna para cada uma das variáveis declaradas no algoritmo e uma coluna para a saída de dados (o que vai ser impresso na tela)

3. Em seguida, acompanhe linha a linha a execução do algoritmo, anotando nas colunas apropriadas cada mudança de valor das variáveis ou do que foi escrito na tela.

4. Preste atenção nas estruturas condicionais (porque pode haver instruções que não serão executadas) e nas estruturas de repetição (porque pode haver trechos de instruções que devem ser executados mais de uma vez).

5. Siga a execução até chegar ao final do algoritmo

Uma animação interessante sobre um exemplo de teste de mesa simples pode ser vista em ***http://www.brasilacademico.com/ed/testemesa.htm.*** *Vamos dar mais um exemplo. Suponha um algoritmo para ler duas notas de um aluno e calcular a média das mesmas. Depois indicar se o aluno foi aprovado (média >= 7) ou não. Suponha também que os valores digitados para as notas serão 8.0 e 9.0, respectivamente.*

```
1  inicio
2       variavel real n1 , n2 , media
3         escrever ( "Digite a sua 1ª nota" )
4       ler n1
5       escrever ( "Digite sua 2ª nota" )
6       ler n2
7       media <- ( n1 + n2 ) / 2
8       se ( media >= 7 ) entao
9           escrever ( "Aprovado sua média é: " )
            escrever media
10      senao
11          escrever ( "Reprovado sua média é: " )
12          escrever media
13              fim se
        fim
14
```

Seguindo os passos que foram explicados do teste de mesa, vai ser criada uma coluna para cada variável do algoritmo e uma coluna para o que vai aparecer na tela. Em seguida, você vai seguindo, linha a linha, passo a passo a execução do algoritmo, com os valores de teste sugeridos e vai preenchendo a tabela criada (vide Tabela 1), até terminar o algoritmo.

Executando	Variáveis			
Nº Linha	**N1**	**N2**	**Media**	**Visualização na tela**
3				Digite a sua 1ª nota
4	8.0			
5	8.0			Digite a sua 2ª nota
6	8.0	9.0		
7			8.5	
9				Aprovado sua média é:
10				8.5

Tabela 1 - Exemplo de teste de mesa

Como as variáveis foram preenchidas corretamente e o resultado impresso na tela está correto, o algoritmo criado está correto.

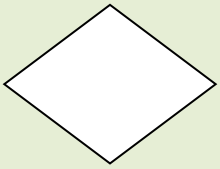
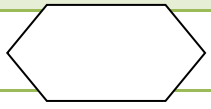
Aula 3 – Fluxogramas

Como foi apresentado anteriormente, o pseudocódigo é uma maneira interessante e bastante utilizada para representar o comportamento das soluções a implementar através de um computador.

Entretanto, uma forma gráfica para a expressão do fluxo de execução de um programa pode apresentar algumas vantagens. O uso de símbolos especiais e a combinação destes símbolos para formar as estruturas mais clássicas de controle, como aquelas apresentadas anteriormente podem eliminar a ambigüidade eventualmente provocada pelo uso do texto escrito.

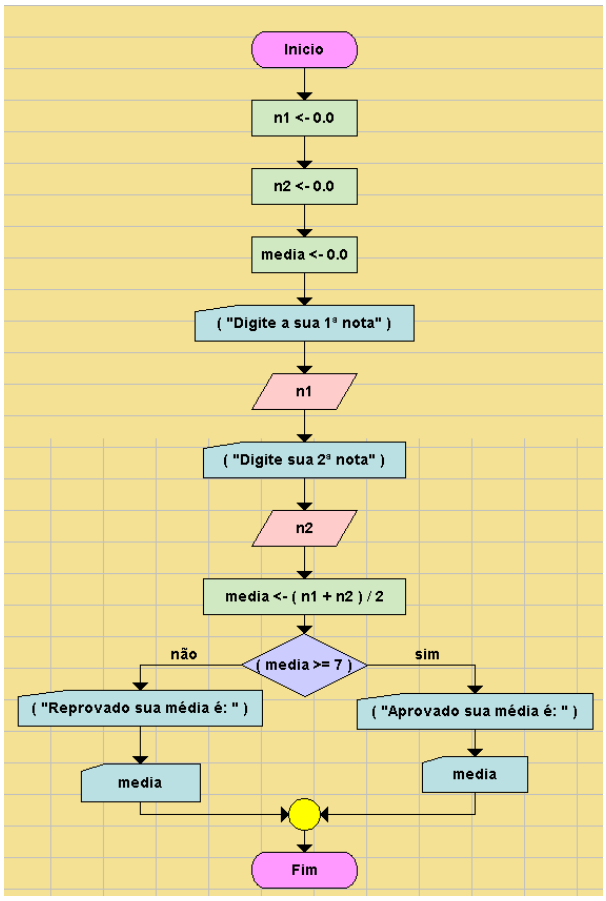
Há muitos anos, o **fluxograma** tem aparecido como uma ferramenta interessante de representação do comportamento de programas, permitindo expressar, além do fluxo lógico da execução e, as operações envolvidas no processamento dos dados e as entradas e saídas. Os fluxogramas são construídos a partir do uso de símbolos padronizados que expressam classes de operações comumente utilizadas nos programas.

Simbologia
Existem diversos símbolos em um diagrama de bloco. No decorrer do curso apresentaremos os mais utilizados. Veja no quadro a seguir alguns dos símbolos que iremos utilizar:

Símbolo	Descrição
	**Terminal:** Representa o início e fim do fluxograma.
	**Processamento:** Representa a execução de operações ou ações como cálculos, atribuições de valores das variáveis, dentre outras.
	**Entrada de Dados:** Representa a entrada de dados para as variáveis através do teclado.
	**Saída de vídeo:** Através deste símbolo podemos representar a saída de informações (dados ou mensagens) por meio de um dispositivo visual de saída de dados, o monitor de vídeo e outros.
	**Decisão:** Representa uma ação lógica, que realizará uma sequência de instruções sendo verdadeiro ou falso, se o teste lógico apresentar o resultado “verdadeiro”, realizará uma sequência se o teste lógico apresentar resultado “false” será executado outra sequência. ***Na aula 10 em estruturas de decisão veremos mais afundo os teste lógicos com implementações em uma linguagem de programação, pseudocódigo e fluxograma.***
	**Preparação:** Representa uma ação de preparação para o processamento, um processamento predefinido, este tipo de

	representação será bastante utilizado na aula 14 onde trataremos de procedimentos e funções.
	**Conector:** Este símbolo é utilizado para interligar partes do fluxograma ou desviar o fluxo para um determinado treco do fluxograma.
	**Conector de Página:** Utilizado para ligar partes do fluxograma em páginas distintas.
	Seta: Orienta a sequencia de execução ou leitura, que poderá ser horizontal ou vertical.

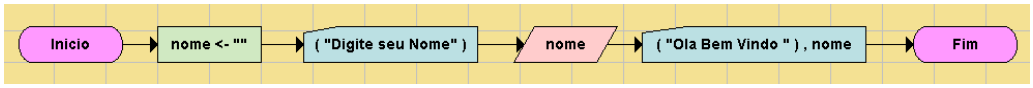
O fluxograma pode ser representado de forma horizontal ou vertical, dentro dos símbolos sempre terá algo escrito, pois somente os símbolos não nos dizem nada. No exemplo abaixo veremos a representação de um algoritmo com o fluxograma e o pseudocódigo:




```
inicio
     variavel real n1 , n2 , media
       escrever ( "Digite a sua 1ª nota" )

    ler n1

    escrever ( "Digite sua 2ª nota" )
    ler n2

    media <- ( n1 + n2 ) / 2

    se ( media >= 7 ) entao

        escrever ( "Aprovado sua média é: " )
        escrever media

    senao
        escrever ( "Reprovado sua média é: " )
        escrever media
            fim se
    fim
```

<= Fluxograma da solução para cálculo da média de uma disciplina, ao lado vemos a mesma solução sendo representada em pseudocódigo, podemos notar que no fluxograma as variáveis são inicializadas com zero, na montagem do fluxograma não é obrigado as variáveis serem inicializadas, se não houver a inicialização no fluxograma o mesmo ainda estará correto e será executado perfeitamente no Portugol ide .

Na imagem abaixo vemos um representação de fluxograma no sentido horizontal.



Representação do algoritmo acima em pseudocódigo.

```
inicio
    variavel texto nome
    escrever ( "Digite seu Nome" )
    ler nome
    escrever ( "Ola Bem Vindo " ) , nome
fim
```

Exercício Prático

1 - Dado os pseudocódigos abaixo, montar o fluxograma equivalente para cada item.

a) Algoritmo para calcular quanto se vai pagar no frete de um determinado produto, sendo que a empresa cobra 1.50 o km e taxa de envio de R$ 9,00.

```
inicio
    variavel real kmrodado , taxa , vlfrete
    escrever ( "Digite os Km rodados" )
    ler kmrodado
    taxa <- 9.00
    vlfrete <- ( kmrodado * 1.50 ) + taxa
    escrever ( "O valor do seu frete é: " ) , vlfrete
fim
```

b) Algoritmo para saber qual o IMC (Índice de Massa Corporal) de uma pessoa.

```
inicio
    variavel real altura , imc , peso
    escrever ( "Digite seu peso" )
    ler peso
    escrever ( "Digite sua altura" )
    ler altura
    imc <- ( peso ) / ( altura * altura )
    escrever ( "Seu Imc é:" ) , imc
fim
```

c) Algoritmo que ler dois valores e efetua as trocas dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A.

```
inicio
    variavel inteiro a , b , troca
    escrever ( "Digite o valor(numérico) da variável A: " )
    ler a
    escrever ( "Digite o valor(numérico) da variável B: " )
    ler b
    troca <- a
    a <- b
    b <- troca
    escrever ( "O novo valor de A é: " ) , a
    escrever ( " O novo valor de B é: " ) , b
fim
```

Aula 4 – Conceitos de Programação

4.1 - Linguagens de Baixo Nível

São linguagens totalmente dependentes da máquina, ou seja, o programa que utiliza este tipo de linguagem não pode ser migrado ou utilizado em outras máquinas. Ao estar praticamente desenhado para aquele hardware, aproveitam ao máximo as características do mesmo.

Dentro deste grupo se encontram:

A LINGUAGEM MÁQUINA: esta linguagem ordena à máquina as operações fundamentais para seu funcionamento. Consiste na combinação de 0's e 1's para formar as ordens entendíveis pelo hardware da máquina.
Esta linguagem é muito mais rápida que as linguagens de alto nível.
A desvantagem é que são bastante difíceis de manejar e usar, além de ter códigos fonte enormes onde encontrar uma falha é quase impossível.

A linguagem Assembler é um derivado da linguagem máquina e está formada por abreviaturas de letras e números chamados mnemotécnicos. Com o aparecimento desta linguagem se criaram os programas tradutores para poder passar os programas escritos em linguagem assembler à linguagem de máquina. Como vantagem com respeito ao código máquina é que os códigos fontes eram mais curtos e os programas criados ocupavam menos memória. As desvantagens desta linguagem continuam sendo praticamente as mesmas que as da linguagem assembler, acrescentando a dificuldade de ter que aprender uma nova linguagem difícil de provar e manter.

4.2 - LINGUAGENS de Alto Nível

São aquelas que se encontram mais próximas à linguagem natural do que à linguagem de máquina. Estão dirigidas a solucionar problemas mediante o uso de EDD's.

> **Nota:** EDD's são as abreviaturas de Estruturas Dinâmicas de Dados, algo muito utilizado em todas as linguagens de programação. São estruturas que podem mudar de tamanho durante a execução do programa. Permitem-nos criar estruturas de dados que se adaptem às necessidades reais de um programa.

Trata-se de linguagens independentes da arquitetura do computador. Sendo assim, a princípio, um programa escrito em uma linguagem de alto nível, pode ser migrado de uma máquina a outra sem nenhum tipo de problema.

Estas linguagens permitem ao programador se esquecer completamente do funcionamento interno da máquina/s para a que está desenhando o programa. Somente necessita de um tradutor que entenda o código fonte como as características da máquina.

Costumam usar tipos de dados para a programação e existem linguagens de propósito geral (qualquer tipo de aplicação) e de propósito específico (como FORTRAN para trabalhos científicos).

4.3 - Linguagens de Médio nível

Trata-se de um termo não aceito por todos, porém certamente vocês já devem ter escutado.

Estas linguagens se encontram em um ponto médio entre as duas anteriores. Dentro destas linguagens poderia se situar C já que pode acessar aos registros do sistema, trabalhar com endereços de memória, todas elas características de linguagens de baixo nível e ao mesmo tempo realizar operações de alto nível.

Gerações

A evolução das linguagens de programação pode ser dividida em cinco etapas ou gerações.

PRIMEIRA GERAÇÃO: Linguagem máquina.

SEGUNDA GERAÇÃO: Criaram-se as primeiras linguagens assembler.

TERCEIRA GERAÇÃO: Criam-se as primeiras linguagens de alto nível. Ex: C, Pascal, Cobol…

QUARTA GERAÇÃO: São linguagens capazes de gerar código por si só, são os chamados RAD, com o qual pode-se realizar aplicações sem ser um expert na linguagem. Aqui também se encontram as linguagens orientadas a objetos, tornando possível a reutilização de partes do código para outros programas. Ex: Visual, Natural Adabes…

QUINTA GERAÇÃO: Aqui se encontram as linguagens orientadas à inteligência artificial. Estas linguagens ainda estão pouco desenvolvidas. Ex: LISP

4.4 - Linguagens de Programação

Uma linguagem de programação é um método padronizado para expressar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sobre várias circunstâncias.

O conjunto de palavras (*tokens*), compostos de acordo com essas regras, constitui o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador.

Uma das principais metas das linguagens de programação é permitir que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos.

Linguagens de programação são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez.

Linguagens de programação também tornam os programas menos dependentes de computadores ou ambientes computacionais específicos (propriedade chamada de portabilidade). Isto acontece porque programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado em vez de ser diretamente executado. Uma meta ambiciosa do Fortran, uma das primeiras linguagens de programação, era esta independência da máquina onde seria executada.

4.5 - TIPOS DE LINGUAGENS

PROGRAMAÇÃO LINEAR

Em matemática, problemas de Programação Linear são problemas de otimização nos quais a função objetivo e as restrições são todas lineares. Muitos problemas práticos em pesquisa operacional podem ser expressos como problemas de programação linear. Certos casos especiais de programação linear, tais como problemas de *network flow* e problemas de *multicommodity flow* são considerados importantes o suficiente para que se tenha gerado muita pesquisa em algoritmos especializados para suas soluções. Vários algoritmos para outros tipos de problemas de otimização funcionam resolvendo problemas de PL como sub-problemas. Historicamente, idéias da programação linear inspiraram muitos dos conceitos centrais de teoria da otimização, tais como dualidade, decomposição, e a importância da convexidade e suas generalizações.

PROGRAMAÇÃO MODULAR

Programação modular é um paradigma de programação no qual o desenvolvimento das rotinas de programação é feito através de módulos, que são interligados entre si através de uma interface comum. Foi apresentado originalmente pela Information & Systems Institute, Inc. no National Symposium on Modular Programming em 1968, com a liderança de Larry Constantine.

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 repetição. Tendo, na prática, sido transformada na Programação modular, a Programação estruturada orienta os programadores para a criação de estruturas simples em seus programas, usando as sub-rotinas e as funções. Foi à forma dominante na criação de software entre a programação linear e a programação orientada por objetos. Apesar de ter sido sucedida pela programação orientada por objetos, pode-se dizer que a programação estruturada ainda é marcantemente influente, uma vez que grande parte das pessoas ainda aprende programação através dela. Porém, a orientação a objetos superou o uso das linguagens estruturadas no mercado.

PROGRAMAÇÃO ORIENTADA A OBJETOS

Orientação a objetos, também conhecida como Programação Orientada a Objetos (POO) ou ainda em inglês *Object-Oriented Programming* (OOP) é um paradigma 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 objetos" 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, e mais natural, para se eliminar o "gap semântico", dificuldade recorrente no processo de modelar o mundo real, no 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. A análise e projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software.

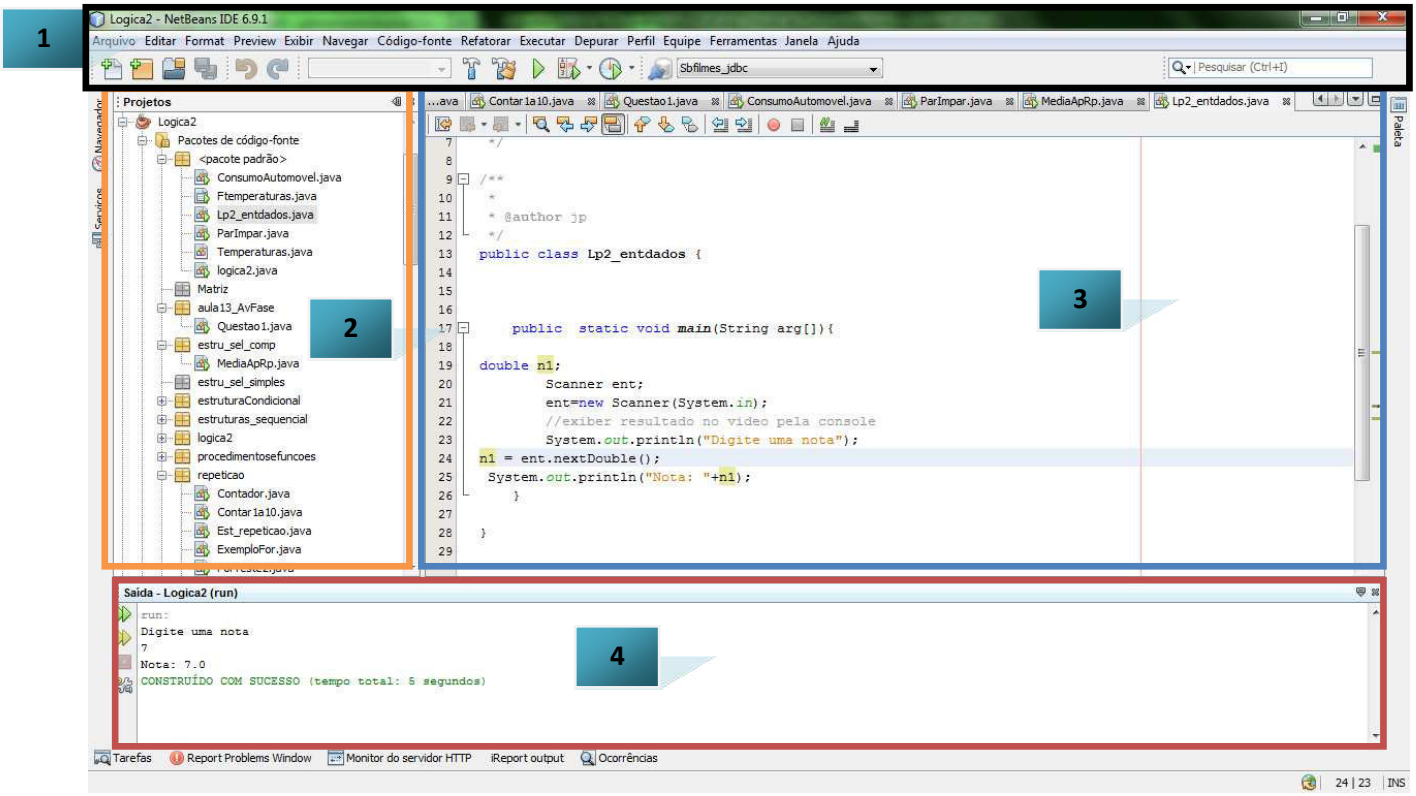
O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos. Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados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

Fase 2 – Construção de Algoritmos

Nesta fase vamos iniciar a construção de algoritmos usando a linguagem de programação Java, para isso vamos conhecer o software que vamos usar o Netbeans versão 6.9.1, no desenvolvimento de softwares chamamos estes softwares usados para desenvolver de IDE (Integrated Development Environment) Ambiente de Desenvolvimento Integrado, através dele podemos desenvolver um aplicativo em uma linguagem de programação ou dependendo da IDE e outras linguagens de programação, nesta etapa iremos construir e transcrever o pseudocódigo criado no Portugol IDE.

Para um melhor entendimento vamos ver como fazemos para receber uma entrada ou saída nesta linguagem de programação e alguns conceitos que mudam quando transcrevemos um pseudocódigo para uma linguagem de programação, então vamos conhecer nossa ferramenta de trabalho

.



Acima temos a imagem de nossa IDE, na tabela abaixo vamos conhecer cada item para aprendermos a utiliza-la, veremos antes alguns ícones importantes para nos ambientarmos a IDE.

 Este ícone representa um projeto Java, dentro dele teremos a organização de todos os algoritmos que iremos criar.

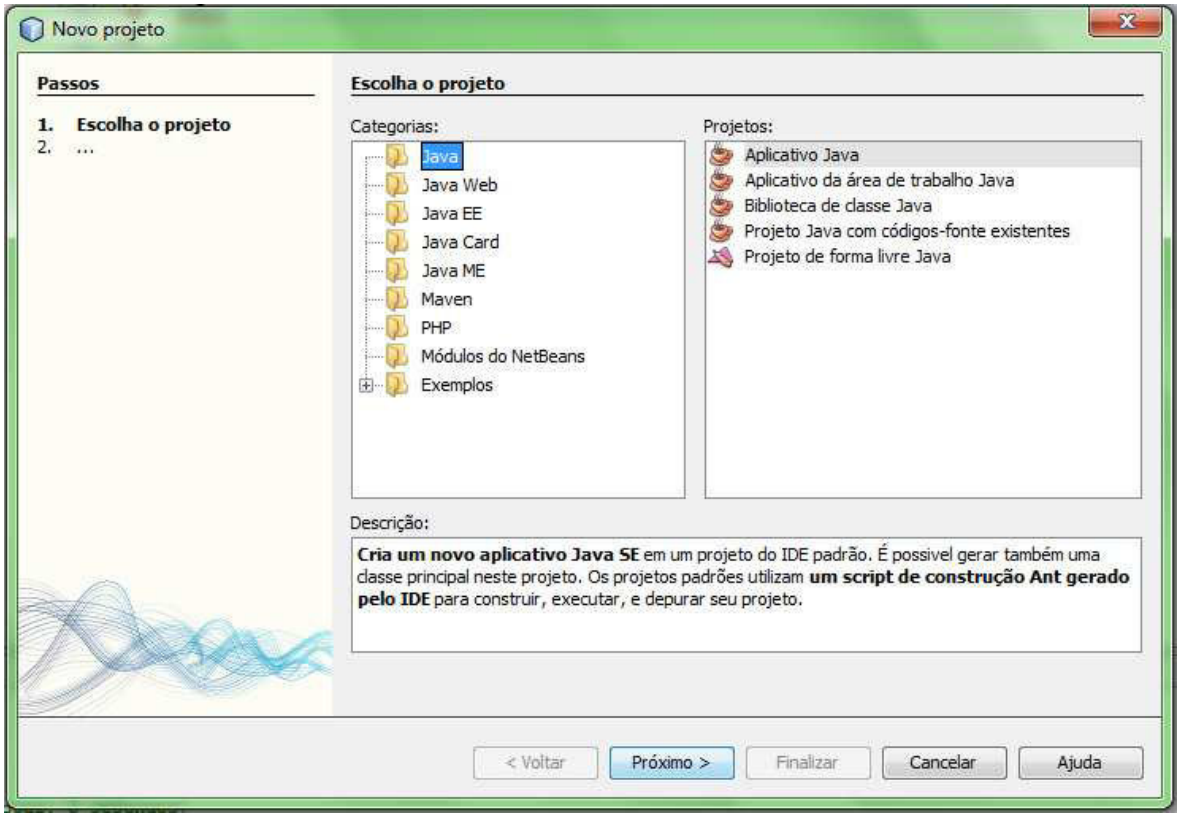
 Este é chamado de pacote, onde nesta etapa podemos compreendê-lo com um diretório, ou uma caixa onde guardamos os nossos arquivos *.java.

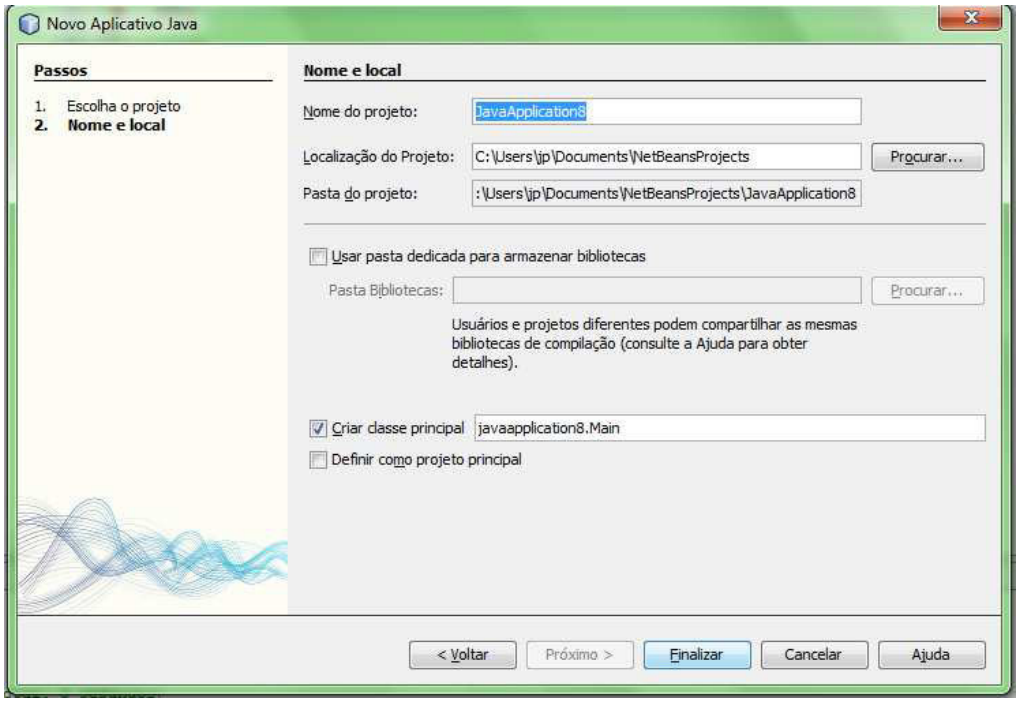
 Com este ícone temos a identificação que este é um arquivo com a extensão *.java.

Item	Descrição
1	Nesta área temos a barra de menu e barra de tarefas, através dela criamos os nossos projetos, como na imagem acima temos o projeto logica2, onde tem várias outras coisa que iremos conhecer.  Com estes ícones da esquerda para a direita, o primeiro podemos criar um novo arquivo que terá a extensão *.java, o segundo criamos o projeto onde teremos todos os nossos arquivos, para cada algoritmo teremos um arquivo que segue as mesmas regras que aprendemos anteriormente sobre nomes de algoritmos, o terceiro ícone utilizamos para abrimos um projeto salvo. O quarto ícone utilizamos para salvar os nossos algoritmos que são os arquivos *.java, onde são chamados de classes, este conceito e definição de classe veremos na próxima disciplina entre outros conceitos de orientação a objetos. Como aprendemos em informática básica em nossa IDE também podemos usar o recurso de desfazer e refazer com estes ícones  Através do menu temos arquivo, Editar, Format e outros que podemos utilizar para realizar as mesmas operações dos ícones citados. Agora como executamos o nosso algoritmo nesta IDE? Usando o menu Executar>Executar arquivo, pelo atalho Shift+F6 ou pelo ícone 
2	Nesta área ficam os projetos e arquivos que foram criados, com o botão direito do mouse em cima do pacote ou projeto na opção novo também podemos criar um arquivo *.java que é denominado classe.
3	Nesta área escrevemos o algoritmo, abaixo veremos uma comparação com o pseudocódigo para entendermos a sintaxe da linguagem.
4	Neste item da imagem é gerada a saída do algoritmo, assim como é no portugol IDE, nesta fase vamos aprender a gerar uma saída gráfica do nosso programa.

Vamos criar um nosso primeiro projeto Java, vamos a arquivo>Novo Projeto ou no ícone de novo projeto como preferir.

Escolha a categoria Java e em projetos Aplicativo Java, depois clique em próximo.



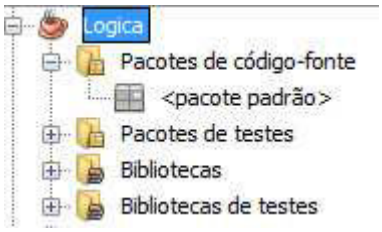


Agora vamos definir o nome do nosso projeto.

Em nome do projeto vamos colocarmos como Logica, em localização do projeto conhecemos onde o projeto será salvo.

Vamos desmarcar a opção criar classe principal, logo mais vamos entender o que é esta classe principal.

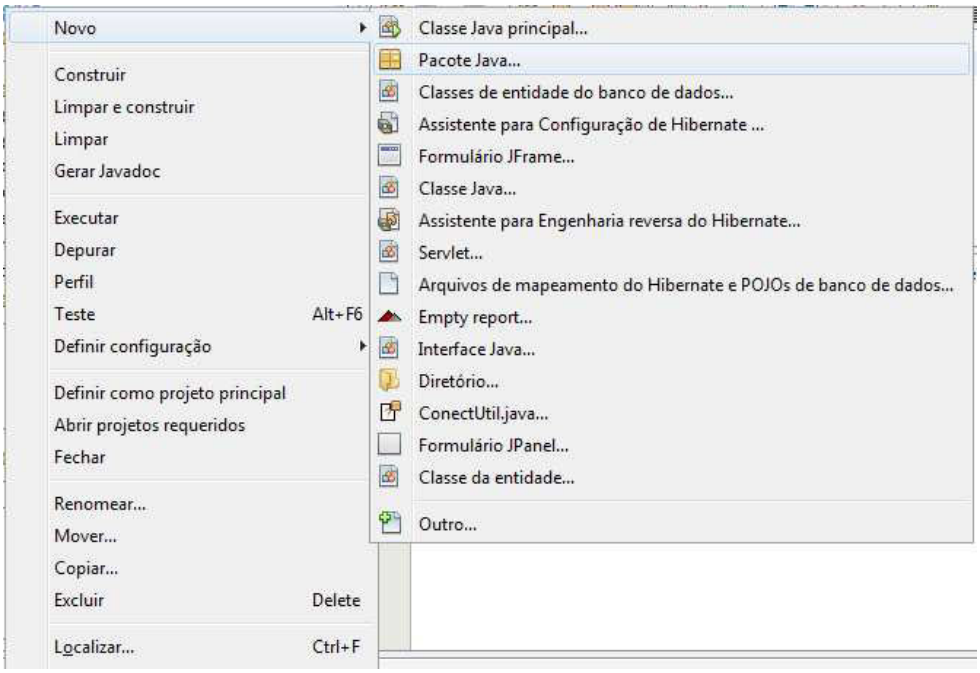
Depois clique em finalizar e o nosso projeto estará criado.

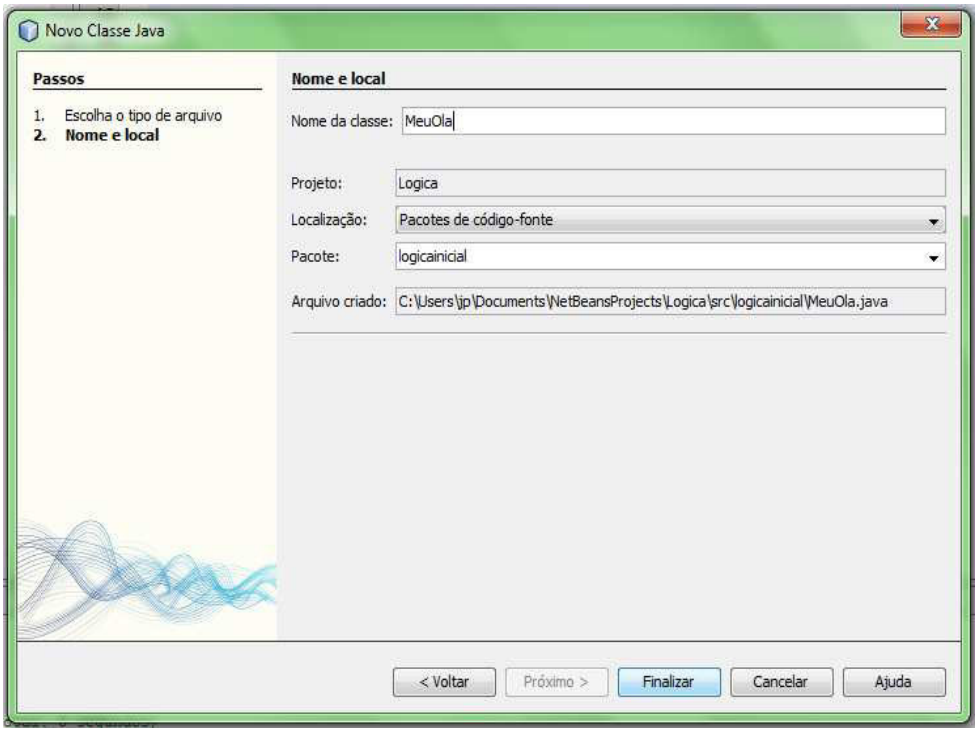


O nosso projeto ficou assim como na imagem ao lado, agora vamos criar um pacote com o nome logicainicial, porque somos profissionais organizados, então a cada aula vamos criar um pacote para melhorar a compreensão dividindo o que já sabemos para depois usarmos.

Com o botão direito em cima do seu projeto escolha a opção novo>Pacote Java para criamos o nosso pacote.

Nesta mesma guia podemos criar o nosso arquivo Java para escrevermos o nosso algoritmo, com o botão direito em cima do pacote criado vamos escolher a opção classe Java.





Vamos chama-la de MeuOla, por questões de padrão e normas adotadas na Linguagem as nossas Classes sempre deverão ter a inicial maiúscula, quando se tratar de uma palavra composta as iniciais de cada palavra devem ser maiúscula.

```
1  /*...*/
5
6  package logicainicial;
7
8  /**
9   *
10  * @author jp
11  */
12 public class MeuOla {
13
14 }
15
```

Após a criação da classe ficará assim como na imagem ao lado, agora vamos a algumas considerações.

No portugol IDE usávamos a pala inicio para começarmos a escrever o algoritmo e fim onde o mesmo terminava, agora usamos as chaves para iniciar e finalizar o algoritmo, logo tudo que iremos codificar estará entre as chaves.

Para que o meu algoritmo possa ser executado e eu possa vê-lo na saída da IDE, é necessário que eu tenha o método main, que faz com que a minha classe seja executada e possa ter interação com o usuário, este método que é uma instrução padrão que devemos criar, e o nosso algoritmo deve estar dentro das chaves deste método, para que possa haver a interação do programa com o usuário, ao lado temos como ficará a estrutura inicial do nosso programa.

```
12 public class MeuOla {
13
14     public static void main(String Args[]){
15
16
17
18     }
19
20 }
```

Nesta área será escrito o nosso algoritmo.

No Portugol IDE quando queríamos escrever algo usávamos a palavra **escrever** agora em nossa linguagem de programação iremos usar a seguinte instrução System.out.println(“Ola mundo”);, tudo que eu quero que seja exibido ao usuário irei colocar dentro dos parênteses e com as aspas duplas, finalizando a linha com ponto e virgula, ficando como na imagem ao lado;

```
12 public class MeuOla {
13
14     public static void main(String Args[]){
15
16 System.out.println("Ola");
17
18
19     }
20
21 }
```

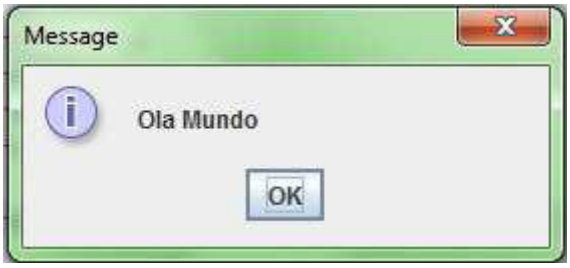
Para executar podemos utilizar o atalho Shift+F6, que irá gerar a saída abaixo;Podemos usar System.out.print("Ola");

System.out.println("Ola");

A diferença entre eles é que o print ele gera a saída do conteúdo na mesma linha, e o println realiza a quebra de linha.

```
Saída - Logica (run)
run:
Ola
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Podemos gerar uma saída de vídeo de forma gráfica conforme a imagem ao lado através do comando ***JOptionPane.showMessageDialog(null, "Ola Mundo");***



Onde é necessário dentro do parêntese passar o valor null e depois da vírgula o que você deseja que seja apresentado no vídeo, para montar esta mensagem gráfica será solicitado um importe para acionar o recurso gráfico, import javax.swing.JOptionPane;

O import deve está localizado acima da declaração da classe, como podemos ver na imagem ao lado, a própria IDE do netbeans vai pedir automaticamente para inserir este import.

```
import javax.swing.JOptionPane;

/**
 *
 * @author jp
 */
public class MeuOla {
```

Aula 5 – Elementos utilizados nos algoritmos

TIPOS DE DADOS, VARIÁVEIS E CONSTANTES.

Tipos podem ser, por exemplo: inteiros, reais, caracteres, etc. As expressões combinam variáveis e constantes para calcular novos valores.

Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado.

Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e às vezes um valor inicial.

5.1 - Tipos de dados

Vamos conhecer os tipos de dados primitivos que podem ser utilizados em Java.

Tipos de dados	Definição	Tipos de dados primitivos da linguagem Java.	Capacidade de armazenamento na memória do computador
Caracteres, literais ou texto.	Específicas para armazenamento de conjunto de caracteres.	char	16 bits unicode
Inteiro	Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos, recebendo números inteiros positivos ou negativos.	byte	8 bits – de (-128) até (127)
		short	16 bits de (-32.768 até 32.767)
		int	32 bits de (-2.147.483.648 até 2.147.483.647)
		long	64 bits de (-9.223.372.036.854.775.808) a (9.223.372.036.854.775.807)
Real	Especificas para armazenar números reais, com casas decimais, positivos ou negativos.	float	32 bits de (-3,4E-38) até (-3,4E+38)
		double	64bits de (-1,7E-308) até (-+1,7E-308)
Lógico	Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso.	boolean	8 bits, onde são armazenados true ou false

Nesta tabela vamos conhecer algumas classes que são utilizadas para armazenar dados, estas não são tipos de dados mais podem ser utilizadas, através delas podemos converter tipos de dados em outros, por exemplo, tenho um tipo de dados primitivo numérico e quero armazenar este valor em um tipo caractere para isso ser possível utilizamos estas classes que veremos na tabela abaixo.

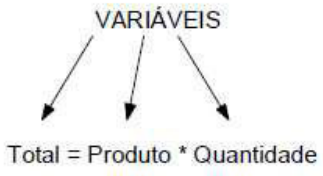
Vamos entender estas classes neste momento como conjunto de recursos que podemos utilizar em nossos programas, mais tarde na disciplina de programação orientada a objetos vamos aprofundar estes conceitos e práticas.

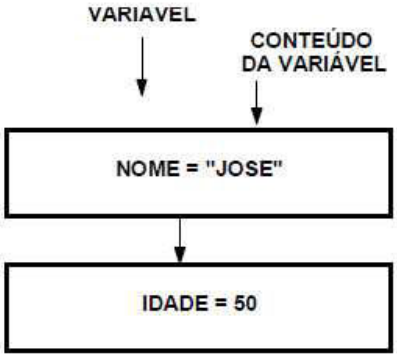
Tipos de dados	Classe	Tipos de dados primitivos em Java
Caracteres, literais ou texto.	String, Character	char
Inteiro	Integer, Byte, Short, Long	byte, short, int, long
Real	Double, Float	double, float
Lógico	Boolean	boolean

5.2 - Variáveis

Variável é a representação simbólica dos elementos de certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

EXEMPLOS DE VARIÁVEIS





TIPOS DE VARIÁVEIS

As variáveis e as constantes podem ser basicamente de quatro tipos: Numéricas, caracteres, alfanuméricas ou lógicas independente da linguagem de programação.

NUMÉRICAS: Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos. Podem ser ainda classificadas como Inteiras ou Reais.

As variáveis do tipo inteiro são para armazenamento de números inteiros e as Reais são para o armazenamento de números que possuam casas decimais.

CARACTERES: Específicas para armazenamento de conjunto de caracteres que não contenham números (literais). Ex: nomes.

ALFANUMÉRICAS: Específicas para dados que contenham letras e/ou números. Pode em determinados momentos conter somente dados numéricos ou somente literais. Se usado somente para armazenamento de números, não poderá ser utilizada para operações matemáticas.

LÓGICAS: Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso.

```
14  public class MeuOla {
15
16      public static void main(String Args[]){
17  char nome='F';
18  String nome2="asdrf5560@4489538!@#$";
19  boolean teste;
20  Integer numero = null;
21  Character caarcte = '5';
22  Byte b;
23  Short sh;
24  Long longo;
25  Boolean ver=true;
26
27      }
28
29  }
```

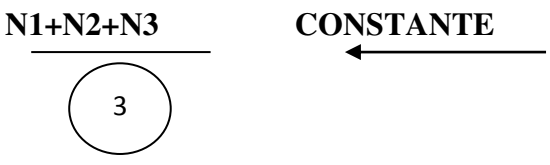
DECLARAÇÃO DE VARIÁVEIS

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas e literais, abaixo veremos como declarar estas variáveis em Java, como vimos no quadro anterior utilizaremos os tipos primitivos e as classes neste exemplo.

Na imagem ao lado podemos observar vários tipos de variáveis com o tipo primitivo ou sendo utilizadas classes, o que devemos usar? Isso vai depender da aplicabilidade do seu algoritmo.

5.3 - Constantes

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.
Exemplo de constantes:



Constantes em Java
Em Java declaramos uma constante da seguinte forma: final double valorfixo=2.45;

Agora que sabemos onde vamos armazenar nossos dados, ainda fica uma pergunta como vamos fazer para receber os dados do usuário?

Entrada de dados

Vamos aprender como fazer o nosso algoritmo receber uma entrada do teclado em Java. Anteriormente em nosso pseudocódigo, quando queríamos receber uma entrada do teclado usávamos a palavra **ler (variável),** agora com a linguagem de programação vamos utilizar a sintaxe abaixo;

```
31  int idade;
32  String nomecompleto;
33  double nota1;
34  Scanner teclado = new Scanner(System.in);
35  idade = teclado.nextInt();
36  nomecompleto = teclado.nextLine();
37  nota1 = teclado.nextDouble();
```

Nas linhas 31 a 33 declaramos as variáveis.
Linha 34 utilizaremos o Scanner(classe) que irá gerar um importe, depois colocamos o nome do objeto que irá receber o entrada do teclado com o nome "teclado" este pode ser nomeado como você quiser, por tanto que seja um nome minúsculo, esta declaração só precisa ser feita uma única vez, como vemos nas linhas 35 a 37 estamos utilizando o mesmo objeto criado.

Para cada tipo de dados eu tenho uma instrução next a ser utilizada, na tabela abaixo iremos ver cada instrução e uma definição.

Instrução (método)	Finalidade
next()	Aguarda uma entrada no formato String (caractere)
nextLine()	Aguarda uma entrada no formato String (caractere)
nextInt()	Aguarda uma entrada no formato Inteiro (numérico)
nextByte()	Aguarda uma entrada no formato Inteiro
nextLong()	Aguarda uma entrada no formato Inteiro Longo
nextDouble()	Aguarda uma entrada no formato numero fracionário.
nextFloat()	Aguarda uma entrada no formato numero fracionário.

Método: Uma ação ou um conjunto de instruções que deverá ser executada.
Objeto: É algo que armazena dentro de si dados ou informações sobre sua estrutura e possui comportamentos definidos por suas operações.
Na disciplina de programação Orientada a objetos estudaremos a fundo estes assuntos.

Entrada de dados com caixa de dialogo

Podemos também realizar a captura de dados do teclado usando caixas de dialogo, usaremos a mesma classe (JOptionPane) que usamos anteriormente para gerar uma saída de vídeo.

Como já sabemos para receber uma entrada do teclado precisamos ter as variáveis, logo o primeiro passo será declarar as mesmas. Vejamos o exemplo abaixo;

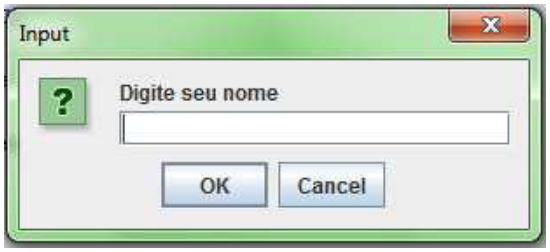
```
13  public class EntradaGrafica {
14
15      public static void main(String[] args) {
16  String nome = null;
17
18  nome = JOptionPane.showInputDialog("Digite seu nome");
19
20  JOptionPane.showMessageDialog(null, nome);
21      }
22
23  }
```

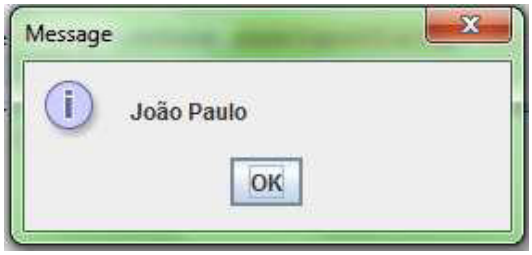
Após termos declarado a variável nome do tipo string.

Temos que dizer que a variável nome (=) recebe o conteúdo de ***JOptionPane.showInputDialog*** e o texto que irá aparecer para o usuário ***("Digite seu nome")*** finalizando com ;.

Depois na linha 20 usamos o ***JOptionPane.showMessageDialog*** onde é passado dentro do parêntese 2 valores o ***(null, nome);*** onde nome é a variável do tipo String que foi declarada na linha 16 para gerar a saída do que foi digitado pelo usuário.

Com isso obtemos o resultado abaixo;





Dessa forma podemos receber as entradas do teclado e gerar uma saída, agora vamos ver como podemos fazer para receber outros tipos de dados usando as caixas de dialogo.

```
13  public class EntradaGrafica {
14          public static void main(String[] args) {
15  String nome = null;
16  int idade;
17  nome = JOptionPane.showInputDialog("Digite seu nome");
18  idade = Integer.parseInt(JOptionPane.showInputDialog("Digite sua idade"));
19
20  JOptionPane.showMessageDialog(null, "Seu nome é "+nome+" sua idade "+idade);
21      }
22  }
```

No exemplo acima temos uma variável do tipo inteira chamada idade, na linha 18 para que o ***JOptionPane.showInputDialog*** possa ser usando com este tipo inteiro precisamos converte-lo pois o JOptionPane.showInputDialog só recebe objetos do tipo string, logo toda vez que precisar realizar uma conversão posso esta utilizando esta sintaxe, se o tipo de dados fosse Double então seria Double.parseDouble e assim de acordo com cada tipo de dados, na tabela abaixo vemos outros tipos de dados e algumas formas de realizar a conversão.

Conversão de tipos de dados	
Integer.parseInt()	***Transformar um inteiro no tipo de dados String***
Double.parseDouble()	***Transformar uma variável double para tipo de dados String.***
Float.parseFloat()	***Transformar um float para tipo de dados String***
Integer.valueOf()	***Você pode também usar o método valueOf() para realizar a conversão.***
Double. valueOf()	
Float. valueOf()	
String.valueOf()	

Aula 6– Operadores

OPERADORES
Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador.

Temos três tipos de operadores:

- ✓ **Operadores de Atribuição**
- ✓ **Operadores Aritméticos**
- ✓ **Operadores Relacionais**
- ✓ **Operadores Lógicos**

6.1 - Operadores de Atribuição

Estes operadores são utilizados para expressar o armazenamento de um valor em uma variável, no qual pode ser variável ou o resultado de um processamento.

Representação em Portugol	**Representação em Java**
<-	=
Exemplo: consumo <- distancia / combustivel	Exemplo: Em Java lê-se media recebe (=) soma/7; media=soma/7;

6.2 - Operadores Aritméticos

Os operadores aritméticos são os utilizados para obter resultados numéricos. Além da adição, subtração, multiplicação e divisão, podem utilizar também o operador para exponenciação.

Os símbolos para os operadores aritméticos são:

Operação	Representação em Portugol	Representação em Java
Adição	+	+
Subtração	-	-
Multiplicação	*	*
Divisão	/	/
Incremento	Utiliza-se a expressão a+1	++
Decremento	Utiliza-se a expressão a-I	--
Exponenciação	^ ou * *	Classe Math Em Java, as operações matemáticas são realizadas utilizando-se métodos da classe Math.
Resto da divisão inteira	% Exemplo: a % b	%

HIERARQUIA DAS OPERAÇÕES ARITMÉTICAS

1 º ( ) Parênteses
2 º Exponenciação
3 º Multiplicação, divisão (o que aparecer primeiro)
4 º + ou – (o que aparecer primeiro)

EXEMPLO

```
TOTAL = PRECO * QUANTIDADE
```

```
1 + 7 * 2 ** 2 –1 = 28
```

```
3 * (1 – 2) + 4 * 2 = 5
```

Portugol

```
inicio
    variavel inteiro a , b
    ler a
    ler b
    escrever ( "Operadores" )
    escrever "\n" , a , " + " , b , " = " , a + b
    escrever "\n" , a , " - " , b , " = " , a - b
    escrever "\n" , a , " * " , b , " = " , a * b
    escrever "\n" , a , " / " , b , " = " , a / b
    escrever "\n" , a , " ^ " , b , " = " , a ^ b
    escrever "\n" , a , " % " , b , " = " , a % b
fim
```

Java

```
18      public static void main(String[] args) {
19       int a , b;
20      a = Integer.parseInt(JOptionPane.showInputDialog("Digite o Valor de A"));
21      b = Integer.parseInt(JOptionPane.showInputDialog("Digite o Valor de B"));
22       System.out.println("Operadores");
23      System.out.println( "\n" + a + " + " + b + " = " + a + b);
24       System.out.println( "\n" + a + " - " + b + " = " +(a - b) );
25       System.out.println( "\n" + a + " * " + b + " = " + a * b);
26       System.out.println( "\n" + a + " / " + b + " = " + a / b);
27       System.out.println( "\n" + a + " expoente " + b + " = " + Math.pow(a, b));
28       System.out.println( "\n" + a + " % " +b + " = " +a % b);
29      }
```

6.3 - Operadores Relacionais

Os operadores relacionais são utilizados para comparar ***String*** de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis. Estes operadores sempre retornam valores lógicos (verdadeiro ou falso/ True ou False). Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses.

Os operadores relacionais são:

Operador	Representação em Portugol	Representação em Java
Igual a	=	==
Diferente de	=/=	!=
Maior que	>	>
Menor que	<	<
Maior ou igual a	>=	>=
Menor ou igual a	<=	<=

EXEMPLO:

Tendo duas variáveis A = 5 e B = 4
Os resultados das expressões seriam:

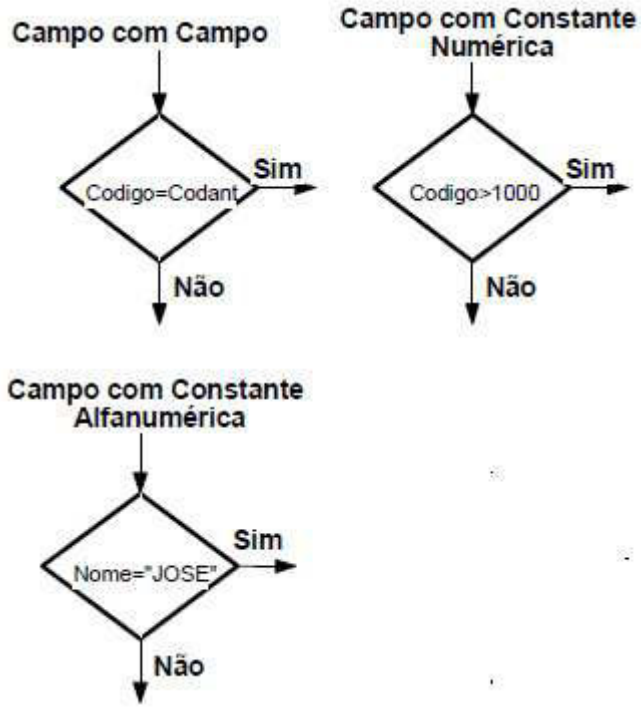
Resultado
5 = 4 = FALSO
5 =/= 4 = VERDADEIRO **5!=4 = VERDADEIRO**
5 > 4 = VERDADEIRO
5 >= 4 =

```
inicio
    variavel inteiro a , b
    ler a
    ler b
    escrever ( "operadores relacionais" )
    escrever "\n" , a , " = " , b , "\t= " , a = b
    escrever "\n" , a , " =/= " , b , "\t= " , a =/= b
    escrever "\n" , a , " > " , b , "\t= " , a > b
    escrever "\n" , a , " >= " , b , "\t= " , a >= b
    escrever "\n" , a , " < " , b , "\t= " , a < b
    escrever "\n" , a , " <= " , b , "\t= " , a <= b
fim
```

VERDADEIRO
5 < 4 = FALSO

```
17  public static void main(String[] args) {
18    Integer a, b;
19    a = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor de A"));
20    b = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor de B"));
21    System.out.println("operadores relacionais");
22    System.out.println("\n" + a + " == " + b + "\t= " + (a == b));
23    System.out.println("\n" + a + " != " + b + "\t= " + (a != b));
24    System.out.println("\n" + a + " > " + b + "\t= " + (a > b));
25    System.out.println("\n" + a + " >= " + b + "\t= " + (a >= b));
26    System.out.println("\n" + a + " < " + b + "\t= " + (a < b));
27     System.out.println("\n" + a + " <= " + b + "\t= " + (a <= b));
28       }
```

Símbolo Utilizado para comparação entre expressões



6.4 - Operadores Lógicos

Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso.

Os operadores lógicos são:

Operador	Descrição	Representação em Portugol	Representação em Java
E (AND)	Uma expressão E (AND) é verdadeira se todas as condições forem Verdadeiras	e	&& ou (& compara bit a bit)
OU (OR)	Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira	ou	ll ou (l compara bit a bit)
NÃO (NOT)	Uma expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.	nao	!

A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos (AND, OR e NOT), T=TRUE e F=FALSE

1º Valor	Operador	2º Valor	Resultado
T	AND	T	T
T	AND	F	F
F	AND	T	F
F	AND	F	F
T	OR	T	T
T	OR	F	T
F	OR	T	T
F	OR	F	F
T	NOT		F
F	NOT		T

EXEMPLOS:

Suponha que temos três variáveis A = 5, B = 8 e C =1

Os resultados das expressões seriam:

Expressões			***Resultado***
A = B	E	B>C	FALSO
A =/= B	OU	B<C	VERDADEIRO
A > B	NAO		VERDADEIRO
A < B	E	B>C	VERDADEIRO
A >= B	OU	B=C	FALSO
A <= B	NAO		FALSO

Portugol

```
inicio
    variavel inteiro a , b , c
    ler a
    ler b
    ler c
    escrever ( "operadores Lógicos" )
    escrever "\n" , a , " = " , b , " e b>c" , "\t= " , a = b e b > c
    escrever "\n" , a , " =/= " , b , " ou b<c" , "\t= " , a =/= b ou b < c
    escrever "\n" , a , " > " , b , " nao" , "\t= " , a > b nao
    escrever "\n" , a , " < " , b , " e b>c" , "\t= " , a < b e b > c
    escrever "\n" , a , " >= " , b , " ou b=c" , "\t= " , a >= b ou b = c
    escrever "\n" , a , " <= " , b , " nao" , "\t= " , a <= b nao
fim
```

JAVA

```
19      public static void main(String[] args) {
20  int a , b , c;
21      a = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor de A"));
22   b = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor de B"));
23    c = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor de C"));
24
25     System.out.println( "operadores Lógicos" );
26     System.out.println( "\n" + a + " == " + b + " e b>c" + "\t= " +(a == b && b > c));
27     System.out.println( "\n" + a + " != " + b + " ou b<c" + "\t= " + (a != b || b < c));
28     System.out.println( "\n" + a + " > " + b + " nao" + "\t= " + !(a > b));
29     System.out.println("\n" + a + " < " + b + " e b>c" + "\t= " + (a < b && b > c));
30     System.out.println( "\n" + a + " >= " + b + " ou b=c" + "\t= " + (a >= b || b == c));
31     System.out.println("\n" + a + " <= " + b + " nao" +"= "+!(a<=b));
32      }
```

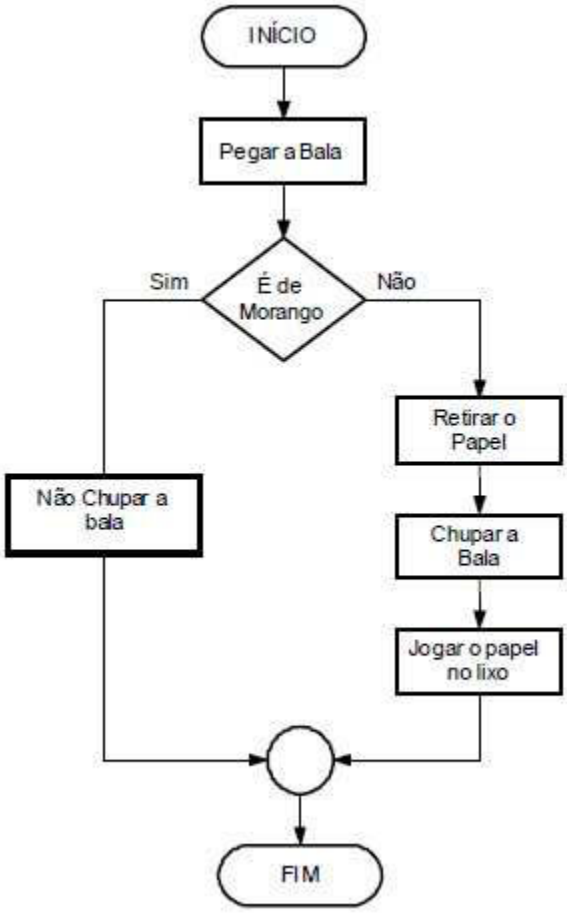
OPERAÇÕES LÓGICAS

Operações Lógicas são utilizadas quando se torna necessário tomar decisões em um diagrama de bloco. Num diagrama de bloco, toda decisão terá sempre como resposta o resultado **VERDADEIRO** ou **FALSO**.

Como no exemplo do algoritmo "CHUPAR UMA BALA". Imaginemos que algumas pessoas não gostem de chupar bala de Morango, neste caso teremos que modificar o algoritmo para:

"Chupar uma bala".
- Pegar a bala
- A bala é de morango?
- Se **sim**, não chupe a bala
- Se **não**, continue com o algoritmo
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

Exemplo: Algoritmo "Chupar Bala" utilizando diagrama de Blocos



Exercício Prático

1- O que é uma constante? Dê dois exemplos.

2 - O que é uma variável? Dê dois exemplos.

3 - Tendo as variáveis SALARIO, IR e SALLIQ, e considerando os valores abaixo. Informe se as expressões são verdadeiras ou falsas.

SALARIO	IR	SALLIQ	EXPRESSÃO	V ou F
100,00	0,00	100	(SALLIQ >= 100,00)	
200,00	10,00	190,00	(SALLIQ < 190,00)	
300,00	15,00	285,00	SALLIQ = SALARIO - IR	

4 - Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são (V) para verdadeiras ou (F) falsas. *Obs.: neste exercício iremos trabalhar com os operadores com a notação do pseudocódigo.*

a) (A+C) > B ( )

b) B >= (A + 2) ( )

c) C = (B –A) ( )

d) (B + A) <= C ( )

e) (C+A) > B ( )

5 – Cite 3 tipos de dados primitivos usados em Java.

6 **–** Elabore um programa em Java que receba duas entradas e retorne o resultado da soma, subtração, multiplicação e divisão. Neste programa vamos usar que tipo de operador?

7 – Com as expressões abaixo crie um programa em Java para informar se as expressões são verdadeiras ou falsas, preencha a tabela 2 com os valores que serão inseridos e na tabela 1 escreva o resultado de cada expressão.

a) (A > C) **AND** (C <= D)

b) (A+B) > 10 **OR** (A+B) == (C+D)

a) (A>=C) **AND** (D >= C)

b) **NAO** (A <= B)

Tabela 1

Itens	***Valores***		
A			
B			
C			
D			

Tabela 2

Resultado das expressões			
a)			
b)			
c)			
d)			

Aula 7 – Vamos a prática – Exercícios

7.1 - Classe Math

Os métodos da classe Math em Java disponibilizam certos cálculos matemáticos comuns, para usar os métodos da classe Math, os métodos da classe Math são static eles são chamados precedendo-se o nome do método com o nome da classe Math e um operador ponto (.). Para gerar uma saída podemos escrever: ***System.out.println(Math.abs(123));*** Que retornará o valor 9.

Na tabela abaixo veremos algumas das funções desta classe e uma descrição da mesma.

Funções da classe Math	Descrição
Math.abs ( );	Retorna o valor absoluto (módulo) do numero passado dentro do parêntese.
Math.acos ( );	Retorna ao usuário o arco-cosseno do angulo passado dentro do parêntese (retorno entre 0 e PI [metade superior de uma circunferência trigonométrica)
Math.asin ( );	Retorna ao arco-seno do angulo passado para o método (retorno entre-PI/2 [3/4] de circunferência trigonométrica.) e PI/2 (1/4 da circunferência).
Math.cos ( );	Retorna o coseno do angulo passado
Math.ceil ( );	Este método retorna o maior numero inteiro (menor que o passado dentro do parêntese)
Math.exp ( );	Retona o valor da Constante de Euller "e" elevada ao numero passado
Math.log ( );	Retorna o logaritmo natural do numero passado.
Math.max ( );	Retorna o maior entre os números passados.
Math.min ( );	Retorna o menor entre os números passados.
Math.pow(... , ...);	Para uma estrutura de potenciação a^b este método retorna o primeiro parâmetro como 'a' e o segundo como 'b'
Math.random ( );	Gera um numero aleatório que vai de zero até um.
Math.sin ( );	Retorna o seno do valor passado dentro do parêntese.
Math.sqrt( );	Retorna a raiz quadrada do numero passado.
Math.tan ( );	Retorna a tangente do ângulo.
Math.toDregrees( );	Retorna o angula passado (em radianos) em graus.
Math.toRadians( );	Retorna o angula passado (em graus) em radianos.

Vamos ver a implementação de algumas funções em pseudocódigo e em Java.

Portugol

```
inicio
    real n1 <- 123.6
    real n2 <- 1.0
    escrever "\nExp (" , n2 , ")\t= " , exp ( n2 )
    escrever "\nABS (" , n1 , ")\t= " , abs ( n1 )
    escrever "\nRaiz (" , n2 * 4 , ")\t= " , raiz ( n2 * 4 )
    escrever "\nLog (" , n2 * 1000 , ")\t= " , log ( n2 * 1000 )
    escrever "\nCos (" , n1 , ")\t= " , cos ( n1 )
fim
```

Java

```
17      public static void main(String[] args) {
18      double n1 =123.6;
19      double n2 = 1.0;
20     System.out.println("\nExp (" + n2 + ")\t= " + Math.exp(n2));
21     System.out.println( "\nABS (" + n1+ ")\t= " + Math.abs(n1));
22     System.out.println( "\nRaiz (" + n1 + ")\t= " + Math.sqrt( n1));
23     System.out.println( "\nLog (" + n2 * 1000 + ")\t= " + Math.log( n2 * 1000 ));
24     System.out.println( "\nCos (" + n1 + ")\t= " + Math.cos( n1 ));
25      }
```

Exercício Prático

01 - Escreva um algoritmo em pseudocódigo e depois o transcreva para Java, o algoritmo deve **LER** um valor (do teclado) e **ESCREVER** (na tela) o seu antecessor.

02 - Escreva um programa em Java para ler as dimensões de um retângulo (base e altura), calcular e escrever a área do retângulo.

03 - Faça um programa em Java que leia idade de uma pessoa expressa em anos, meses e dias e escrever a idade dessa pessoa expressa apenas em dias. Considerar ano com 365 dias e mês com 30 dias.

04 - Com o pseudocódigo abaixo transcreva para uma linguagem de programação Java.

```
inicio
    variavel real kmrodado , taxa , vlfrete
    escrever ( "Digite os Km rodados" )
    ler kmrodado
    taxa <- 9.00
    vlfrete <- ( kmrodado * 1.50 ) + taxa
    escrever ( "O valor do seu frete é: " ) , vlfrete
fim
```

05 - Qual a sintaxe em Java responsável por gerar uma saída de dados no modo gráfico e no prompt.

06 - Qual a sintaxe em Java responsável por gerar uma entrada de dados no modo gráfico e no prompt.

07 - Faça um programa em Java que leia dois números e retorne o maior valor e o menor dos mesmos.

08 - Faça um programa em Java que leia um número e retorne sua raiz quadrada.

09 - Com o código Java abaixo elabore a representação gráfica através de um fluxograma do mesmo.

```
13  public class CalcImc {
14
15      public static void main(String[] args) {
16
17          double altura, imc, peso;
18
19          Scanner entrada = new Scanner(System.in);
20          System.out.println("Digite o seu Peso");
21          peso = entrada.nextDouble();
22
23          System.out.println("Digite a sua altura");
24          altura = entrada.nextDouble();
25          imc = (peso) / (altura * altura);
26
27          System.out.println("Seu IMC é: " + imc);
28
29      }
30  }
```

10 - A loja ELETROMOVEIS esta vendendo os seus produtos no cartão em 5 vezes sem juros, Faça um programa em Java que receba o valor da compra e mostre o valor das parcelas.

11 - Para enviar uma mensagem por fax, um comerciante cobra uma taxa fixa de R$ 1,25 mais R$ 0,55 por página enviada, completa ou não. Qual é o numero mínimo de Páginas que devem ser enviadas para que o preço ultrapasse R$ 10,00?

a) Faça um programa em Java para descobrir qual o mínimo de páginas com o valor determinado.

b) Faça um programa para este comerciante, que ao informar quantas páginas enviadas pelo fax, mostrar o valor que deve ser pago pelo cliente.

12 - Um ciclista percorreu 135 quilômetros em três horas: na primeira hora percorreu 5 quilômetros a mais que na segunda; na terceira hora percorreu a terça parte da distância percorrida na primeira. Quantos quilômetros esse ciclista percorreu na terceira hora? Para ajudar o ciclista vamos desenvolver um programa para saber quantos quilômetros ele percorreu na terceira hora.

Estruturas de Controle

Ao construir uma aplicação, temos a necessidade de controlar o fluxo do sistema, blocos de instruções para solucionar um determinado problema.

Essa necessidade de controle deve-se ao fato de o fluxo poder se repetir ou em determinadas circunstancias nem mesmo precisar ser executado. Para isso temos as estruturas de controle que podem nos dar repetições simples, repetições condicionais e desvio de fluxo.

Na Programação Estruturada utilizam-se três formas básicas para controlar o fluxo das operações: Sequência, Seleção e Repetição, que serão descritas e exemplificadas nas subseções seguintes.

A combinação destas estruturas permite construir algoritmos diversos, em vários níveis de complexidade.

Em Java existem alguns comandos para efetuar controle de fluxo que são:
for
while
do while
if else
switch
break
continue
try
catch
finally

Aula 8 – Estrutura sequencial

Estrutura sequencial é um conjunto de ações executadas numa ordem linear, de cima para baixo, ou da esquerda para a direita, na mesma sequência em que foram escritas.

Quando se tem uma sequência de ações (instruções) que devem ser executadas em conjunto, quer seja dentro de outra estrutura, ou sequencialmente no algoritmo, tem se um bloco de comandos.

No exemplo abaixo temos um programa em Java que lê o nome e a idade de uma pessoa, como resultado no vídeo apresenta o nome digitado e sua idade, nas aulas anteriores fizemos algumas dessas estruturas aqui aprendemos que esses conjuntos de instruções são tecnicamente chamados de estrutura sequencial.

Exemplo:

```
18          int idade;
19         String nome;
20
21    nome=JOptionPane.showInputDialog("Digite seu nome");
22    idade = Integer.parseInt(JOptionPane.showInputDialog("Qual a sua idade"));
23    JOptionPane.showMessageDialog(null,"Obrigado por informar seus dados" + nome + idade);
24
```

Exercício Prático

1 - Escreva um programa em Java para ler a o número total de eleitores de um município, o numero de votos brancos, nulos e válidos. Calcular e escrever o percentual que cada um representa em relação ao total de eleitores.

02 - Uma revendedora de carros usados paga a seus funcionários vendedores um salário fixo por mês, mas uma comissão também fixa para cada carro vendido e mais 5% do valor das vendas por ele efetuadas. Escreva um algoritmo que leia o numero de carros por ele vendidos, o valor total de suas vendas, o salário fixo e o valor que ele receber por carro vendido. Faça um programa em Java que calcule e escreva o salário final do vendedor.

03 - Escreva um programa em Java para ler uma temperatura em graus Fahrenheit, calcular e escrever o valor correspondente em graus Celsius (baseado na formula abaixo):

C/5 = F-32/9

Obs.: Saiba que 100 Graus Celsius correspondem a 212F.

Aula 9 – Estrutura de seleção

Estas estruturas permitem direcionar o fluxo das ações conforme uma condição estabelecida, ou seja, executam as ações de um determinado bloco se a condição de teste retornar um valor Verdadeiro.

Caso a condição retorne Falso, o programa desvia seu fluxo de execução para o primeiro comando após o final da estrutura em questão.

Pode-se também utilizar uma estrutura composta, na qual se adiciona o senão (*else*) imediatamente após o bloco de comandos do se (*if*). O bloco de comandos do senão será executado se, e somente se, a condição de teste do se retornar um valor Falso.

9.1 - Estruturas de Decisão ou Seleção

Como vimos no capítulo anterior em "Operações Lógicas", verificamos que na maioria das vezes necessitamos tomar decisões no andamento do algoritmo. Essas decisões interferem diretamente no andamento do programa. Trabalharemos com dois tipos de estrutura. A estrutura de Decisão simples e composta, encadeada e múltipla escolha.

Comandos de Decisão

Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores.

As principais estruturas de decisão são: "***Se Então***", "***Se então Senão***" e "***Caso Selecione***".

9.1.1 - SE ENTÃO / IF (Estrutura de seleção simples)

A estrutura de decisão "SE/IF" normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando SE/IF então execute determinado comando.

Em Java a estrutura se então, tem a sintaxe abaixo, onde o conjunto de instruções deve ser delimitado por chaves, quando a condição for verdadeira ele irá executar o conjunto de instruções.

```
IF (condição){

Conjunto de instruções

}
```

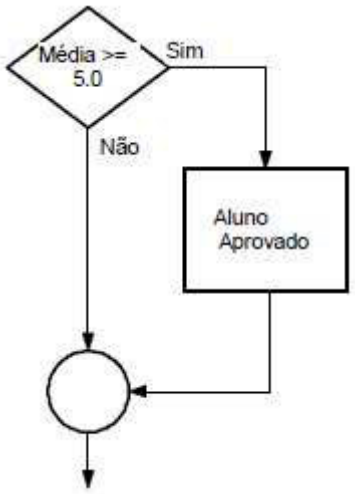
Em Portugol a sintaxe abaixo;

```
se [condição] então
   [instruções]
fimSe
```

Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo como ficaria, neste exemplo esta sendo exibido somente o trecho do código do se, mais a frente veremos o código completo.

SE MEDIA >= 5.0
ENTÃO ALUNO APROVADO

Em diagrama de blocos ficaria assim:



EM JAVA

```
28  String msg = null;
29   double media = 0;
30         if (media >= 5){
31          msg = " Aluno Aprovado ";
32         JOptionPane.showMessageDialog(null,msg);
33         }
```

Em Portugol

```
se media  >=5 entao
escrever ("Aluno Aprovado")
fimse
```

9.1.2 - SE ENTÃO SENÃO / IF ... ELSE (Estrutura de Seleção Composta)

A estrutura de decisão “SE/ENTÃO/SENÃO”, funciona exatamente como a estrutura “SE”, com apenas uma diferença, em “SE” somente podemos executar comandos caso a condição seja verdadeira, diferente de “SE/SENÃO”, pois sempre um comando será executado independente da condição, ou seja, caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executado.

Vejamos a sintaxe;

IF (condição){

Conjunto de instruções

}else{

Conjunto de instruções

}

Em Portugol a sintaxe abaixo;

se [condição] então
[instruções A]

senao

[instruções B]

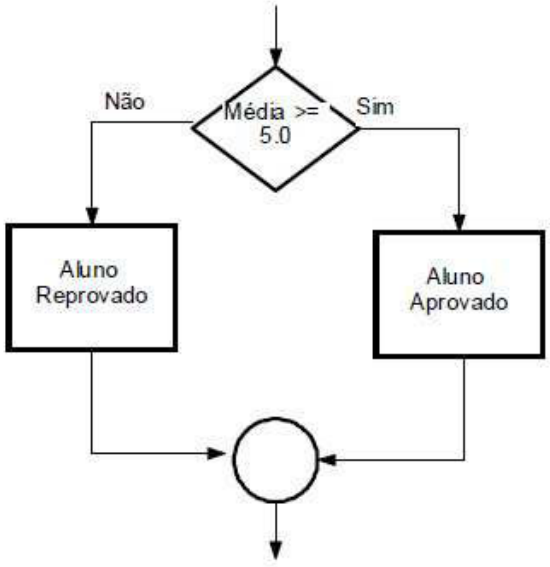
fimSe

Em algoritmo ficaria assim:

SE MÉDIA >= 5.0 ENTÃO
ALUNO APROVADO
SENÃO
ALUNO REPROVADO
FIMSE

```
se ( media >= 5 ) entao
    escrever ( "Aluno Aprovado" )
senao
    escrever ( "Aluno Reprovado" )
fim se
```

Em diagrama



EM JAVA

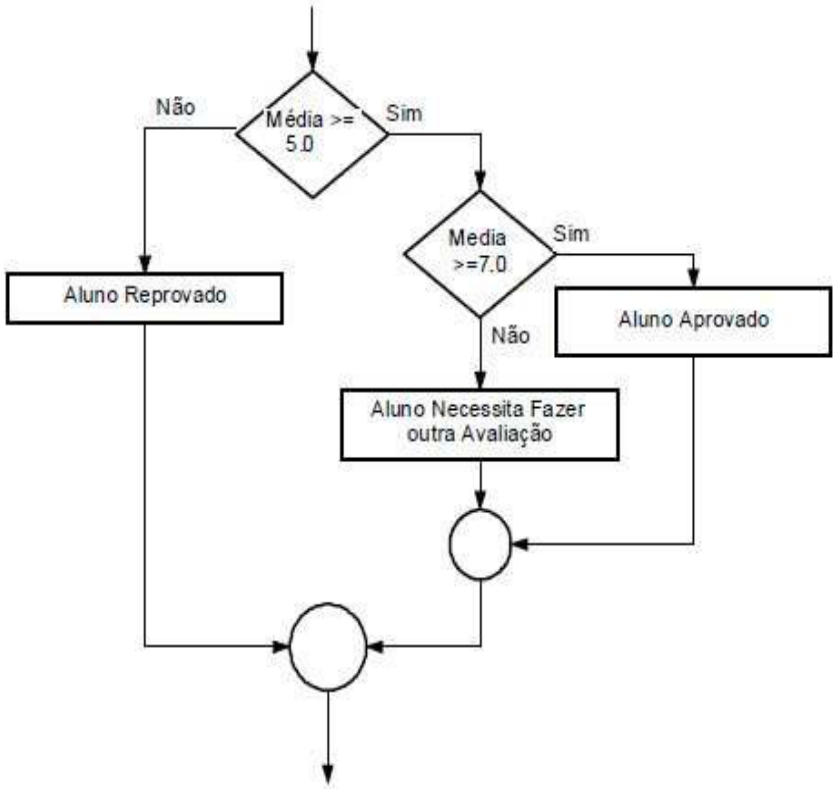
```
29          if (media >= 5){
30           msg = " Aluno Aprovado ";
31          }else{
32          msg = " Aluno Reprovado ";
33          }
34
35  JOptionPane.showMessageDialog(null,msg);
```

No exemplo acima está sendo executada uma condição que, se for verdadeira, executa o comando “APROVADO”, caso contrário executa o segundo comando “REPROVADO”.

9.1.3 - Estrutura de seleção encadeada

É uma sequencia de testes de seleção, devido à necessidade de processamento, agrupamos várias seleções, formaremos uma seleção encadeada. Dentro de uma mesma condição testar outras condições. Como no exemplo abaixo:

```
   media <- ( n1 + n2 ) / 2

se ( media >= 5 ) entao

  se ( media >= 7 ) entao

      escrever ( "Aluno Aprovado" )
    senao
      escrever ( "Aluno Necessita fazer outra Avaliação" )
  fim se
   senao
       escrever ( "Aluno Reprovado" )
   fim se
```



Em Java

```
26          if (media >= 5) {
27
28          if (media>=7){
29  JOptionPane.showMessageDialog(null, "Aluno APROVADO");
30          }else{
31          JOptionPane.showMessageDialog(null, "Aluno Necessita fazer outra Avaliação");
32          }
33
34          }else{
35          JOptionPane.showMessageDialog(null, "Aluno Reprovado");
36          }
```

Vejamos abaixo como ficaria o programa completo, recebendo as variáveis.

```
19      public static void main(String[] args) {
20  double media = 0,n1,n2;
21  n1=Double.parseDouble(JOptionPane.showInputDialog("Digite sua 1° nota"));
22  n2=Double.parseDouble(JOptionPane.showInputDialog("Digite sua 2° nota"));
23  media =(n1+n2)/2;
24
25
26          if (media >= 5) {
27
28          if (media>=7){
29  JOptionPane.showMessageDialog(null, "Aluno APROVADO");
30          }else{
31          JOptionPane.showMessageDialog(null, "Aluno Necessita fazer outra Avaliação");
32          }
33
34          }else{
35          JOptionPane.showMessageDialog(null, "Aluno Reprovado");
36          }
37
38
39      }
```

9.2 - CASO SELECIONE / SWITCH ... CASE – Estrutura de seleção múltipla escolha

A estrutura de decisão CASO/SELECIONE é utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula “Caso”.

Sintaxe em pseudocódigo no Portugol IDE.

```
escolhe [expressão]
      caso [expressão]:
         [instruções]
      caso [expressão]:
         [instruções]
       .....
      defeito :
         [instruções]
   fimescolhe
```

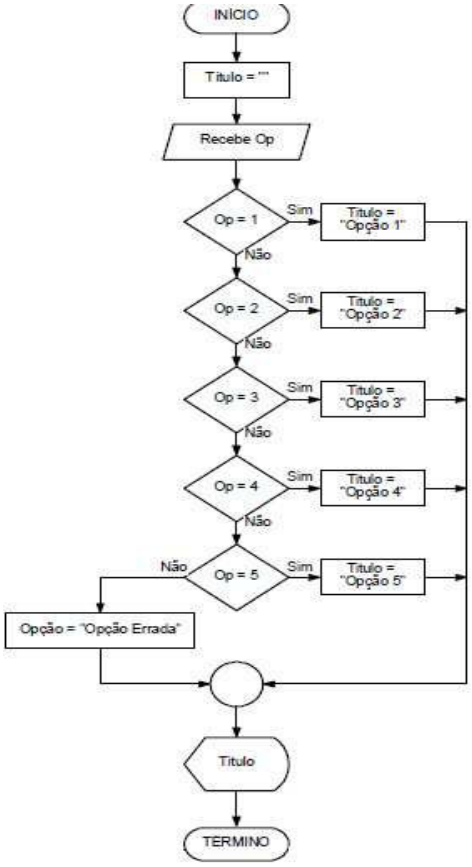
Sintaxe em Java.

```
  switch (expressão)
      {
         case 1: [expressão];
[instruções]
         break;
         case 2: [expressão];
[instruções]
```

```
        break;
        case 3: [expressão];
[instruções]
        break;
      .....
        default: [instrução]

    }
```

No exemplo do diagrama de blocos abaixo, é recebido uma variável "**Op**" e testado seu conteúdo, caso uma das condições seja satisfeita, é atribuído para a variável Titulo a String "Opção X", caso contrário é atribuído à string "Opção Errada".

Veremos em Java e em pseudocódigo um exemplo de utilização desta estrutura, utilizaremos a seguinte sequencia de comandos abaixo representados no diagrama.



Em Portugol

```
inicio
    variavel inteiro op
    variavel texto titulo
    ler op
    escolhe op
        caso 1:
            titulo <- "Titulo 1"
            escrever titulo
        caso 2:
            titulo <- "Titulo 2"
            escrever titulo
        caso 3:
            titulo <- "Titulo 3"
            escrever titulo
        caso 4:
            titulo <- "Titulo 4"
            escrever titulo
        caso 5:
            titulo <- "Titulo 5"
            escrever titulo
        defeito :
            titulo <- "Opção errada"
            escrever titulo
    fimescolhe
fim
```

Em Java.

```
17          public static void main(String[] args) {
18          int op = 0;
19          String titulo;
20          Scanner ent = new Scanner(System.in);
21      op=ent.nextInt();
22          switch (op)
23          {
24              case 1: titulo="Opção 1";
25              System.out.println("Opção escolhida: "+titulo);
26              break;
27              case 2: titulo="Opção 2";
28              System.out.println("Opção escolhida: "+titulo);
29              break;
30              case 3: titulo="Opção 3";
31              JOptionPane.showMessageDialog(null,"Opção escolhida: "+titulo);
32              break;
33              case 4: titulo="Opção 4";
34              JOptionPane.showMessageDialog(null,"Opção escolhida: "+titulo);
35              break;
36              case 5: titulo="Opção 5";
37              JOptionPane.showMessageDialog(null,"Opção escolhida: "+titulo);
38              break;
39              default: System.out.println("Opção Errada");
40          }
41      }
```

Exercício Prático

Executar as tarefas abaixo em pseudocódigo e em Java.

1 - Elabore um programa em Java que leia um número. Se positivo armazene-o em A, se for negativo, em B. No final mostrar o resultado.

2 - Faça um programa que calcula a média de três notas de um aluno, e diga o conceito conforme a tabela abaixo;

Média	Conceito
0 até 4,9	D
5 até 6,9	C
7 até 8,9	B
9 até 10	A

3 - Ler um número e verificar se ele é par ou ímpar. Quando for par armazenar esse valor em P e quando for ímpar armazená-lo em I. Exibir P e I no final do processamento.

4 - Construa um programa em Java para ler uma variável numérica N e imprimi-la somente se a mesma for maior que 100, caso contrário imprime-la com o valor zero.

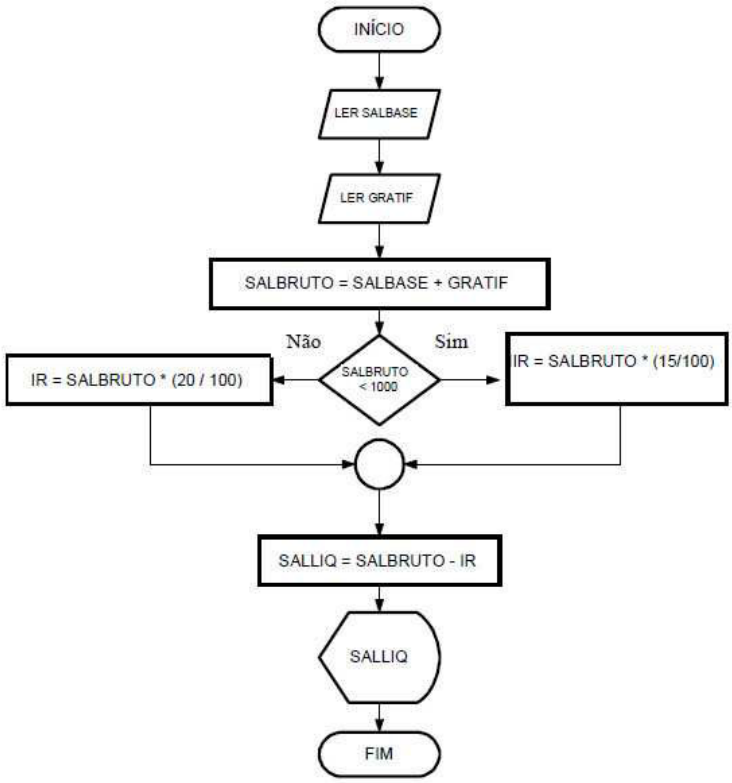
5 - Faça um programa que controle a emissão de Xérox da escola. O programa receberá a quantidade de cópias tiradas, e o tipo de cliente, e com base na tabela abaixo, o programa deverá mostrar como resposta o valor a pagar.

Tipo Cliente	Código	Unitario
Aluno	10	0,15
Professor	12	0,07
Direção	1	0,00
Não Aluno	15	0,20

6 - Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um algoritmo em Java que calcule seu peso ideal, utilizando as seguintes fórmulas:

Para homens: (72.7*h) - 58
Para mulheres: (62.1*h) - 44.7 (h = altura)

7 - Desenvolva um programa em Java do diagrama apresentado abaixo, de acordo com os dados fornecidos:



TESTE O DIAGRAMA COM OS DADOS ABAIXO

SALBASE	GRATIF
3.000,00	1.200,00
1.200,00	400,00
500,00	100,00

MEMÓRIA

SALBASE	GRATIF	SALBRUTO	IR	SALLIQ

DADOS DE SAÍDA

SALLIQ

Elabore um algoritmo levando-se em conta o diagrama apresentado:

1 Elabore um programa em Java que leia o código de uma categoria de produtos e exibir o seu nome conforme a tabela abaixo;

Código da categoria	**Nome da categoria**
1	Produtos de Limpeza
2	Produtos Alimentícios
3	Produtos de consumo

2 Faça um programa que receba a quantidade de litros a abastecer, o tipo de combustível e diga o valor a pagar, conforme tabela:

Combustível	**Valor por litro**
Gasolina	2,79
Álcool	1,99
GNV	1,52
Diesel	1,89

Aula 10 – Estruturas de repetição

COMANDOS DE REPETIÇÃO

Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executado um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Trabalharemos com modelos de comandos de repetição:

Portugol
Enquanto
Para
Repete
Faz

Java
While
For
Do While

10.1 – Estruturas de repetição Enquanto - While

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Vejamos a sintaxe no Portugol e em Java

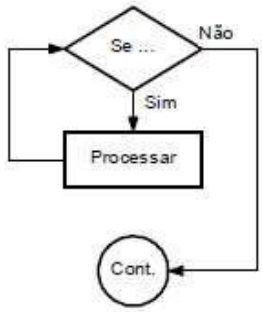
- **Portugol**

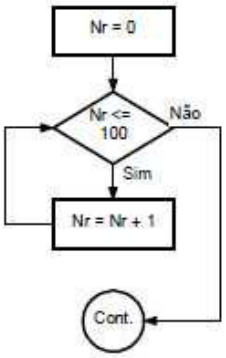
```
enquanto [condição] faz
   [intruções]
fimEnquanto
```

- **Java**

```
while (condição)
{
Conjunto de instruções
}
```

Em diagrama de bloco a estrutura é a seguinte:





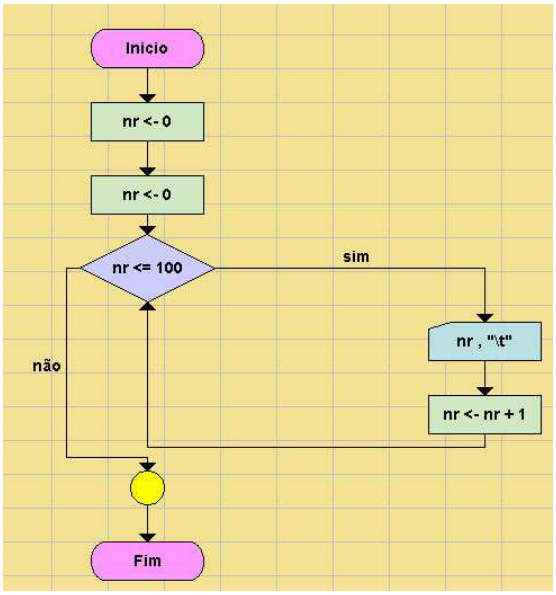
Agora vamos ver um exemplo desta estrutura sendo implementado em Portugol, Fluxograma (Diagrama de bloco) e em Java, enquanto a condição x for verdadeira o conjunto de instruções será executado.

Portugol

```
inicio
    variavel inteiro nr
    nr <- 0
    enquanto nr <= 100 faz
        escrever nr , "\t"
        nr <- nr + 1
    fimenquanto
fim
```

Enquanto a variável inteira **nr** for menor ou igual a 100, escrever nr igual à **nr** mais um.
Neste exemplo implementamos um contador que ira iniciar em 0 e vai ate 100.

Fluxograma (Diagrama de bloco)



Em Java

```
18  public static void main(String[] args) {
19      int nr = -1;
20        while (nr <100) {
21          nr = nr+1  ;
22          System.out.println("Numero " + nr);
23         }
24      }
```

Neste exemplo em Java iniciamos a variável **nr** com **-1** para que seja exibindo o numero **0** e a condição é **nr<100** para que termine a contagem em **100**, podemos também usar no lugar de **nr=nr+1** para **nr=++nr**, onde o ++ é um operador de incremento logo vai incrementar de um em um a cada loop.

10.2- Estrutura de repetição Repete

Repete as instruções até que a condição seja verdadeira, abaixo temos a sintaxe em Portugol, em Java iremos utilizar o while.

Vejamos a sintaxe:

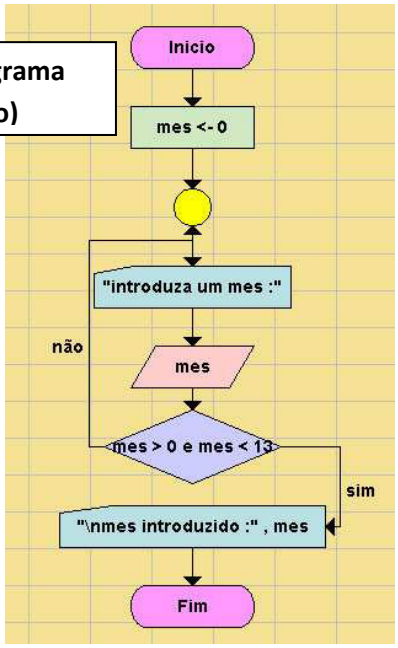
- **Portugol**

repete
[instruções]
até[condição]

Exemplos

```
inicio
    variavel inteiro mes
    repete
        escrever "introduza um mes :"
        ler mes
    ate mes > 0 e mes < 13
    escrever "\nmes introduzido :" , mes
fim
```

Exemplo no Fluxograma (Diagrama de bloco)



Neste exemplo a cima as instruções escrever e ler serão repetidas ate que a condição (**mes > 0 e mes < 13**) seja verdadeira.

Vamos ver como esta estrutura Repete poderia ser implementada em Java usando o while.

```
44    int mes = 14;
45   while (mes==0 ||mes!=0 && mes>13){
46   mes=Integer.parseInt(JOptionPane.showInputDialog("Digite um Mês"));
47    JOptionPane.showMessageDialog(null,"\nMês Introduzido :"+ mes);
48   }
```

A problemática desta situação proposta era que, o programa recebesse um valor numérico para saber se o valor recebido era correspondente a um mês, se não for ele continua pedindo para digitar um mês.
No código Java acima, inicializamos a variável inteira **mes** com o valor 14, e no while a condição **(mês==0 ||mês!= && mês>13)** então enquanto a condição anterior for verdadeira ele executa o bloco de instruções.

O valor inicial de **mes** é 14 que não é igual a 0 então **(falso)** ou mês diferente de 0, o valor 14 é diferente de 0 e maior que 13 por isso ele entra no laço**(verdadeiro)**. Como não existe o mês 0, quando o valor de mes for 0, onde mes=0 é igual a mês==0 com a primeira

condição ele entra no laço porem com a segunda não, mais na primeira condição é mês==0 **ou** é (diferente)!=0, se uma das condições for verdadeira ele entra no laço.

Com estas condições enquanto o usuário digitar valores fora do intervalo dos meses do ano ele continua pedindo para digitar um mês.

Podemos resolver este problema de outras formas, usamos o while para demonstrar também o uso de operadores relacionais e lógicos.

10.3- Estrutura de Repetição Faz – Do...While

Repete as instruções até que a condição seja falsa, abaixo temos a sintaxe em Portugol, em Java iremos utilizar o **do while**.
Vejamos a sintaxe:

Em Portugol

Faz
[instruções]
Enquanto[condição]

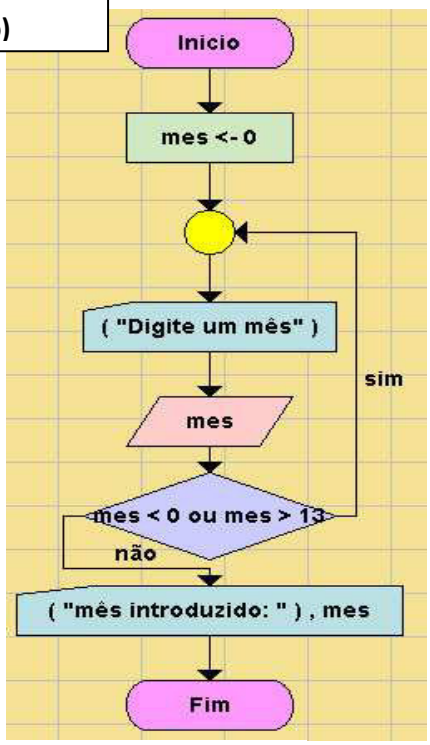
Em Java
do
{
instruções
}while(condição)

Exemplos: Solicita que seja digitado um mês até o utilizador introduzir um valor válido (entre 1 e 12)

Em Portugol

```
inicio
    variavel inteiro mes
    faz
        escrever ( "Digite um mês" )
        ler mes
    enquanto mes < 0 ou mes > 13
    escrever ( "mês introduzido: " ) , mes
fim
```

Exemplo no Fluxograma (Diagrama de bloco)



Com esta estrutura de repetição executamos as instruções até que a condição seja falsa, quando ela for verdadeira a execução do programa para.

Em Java

```
32        int mes=0;
33   do {
34   mes=Integer.parseInt(JOptionPane.showInputDialog("Digite um Mês"));
35    JOptionPane.showMessageDialog(null,"\nMês Introduzido :"+ mes);
36   }while(mes<=0 || mes>=13);
```

Neste exemplo em Java usamos os operadores <= e >= para evitar que seja inserido o valor 0 e 13, se usarmos só o < e > quando o valor for 0 ou 13 a condição fica verdadeira e consequentemente para a execução do programa.

10.4 – Estruturas de Repetição Para - For

Neste caso, a estrutura de repetição **para** se utiliza de variáveis que define o numero de vezes que o conjunto de instruções será executado.

Vejamos a sintaxe:

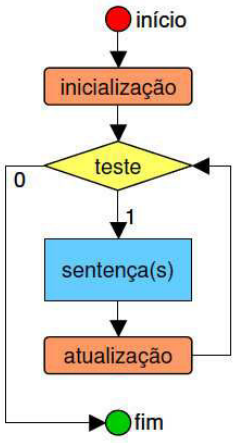
- **Em Portugol**

para [variável numérica] de [valor inicial] ate [valor final] passo [valor de separação]
[instruções]
próximo

Ou

para [variável numérica] de [valor inicial] ate [valor final]
[instruções]
próximo

Estrutura do Fluxograma Para



Em Java

for (variável = valor inicial;condição;incremento)
{
Bloco de instruções
}

No caso do for o valor inicial e o valor final pode ser substituído por variáveis.

Veremos os exemplos de implementação desta estrutura, para ilustrar esse exemplo vamos criar um contador de 0 a 10.

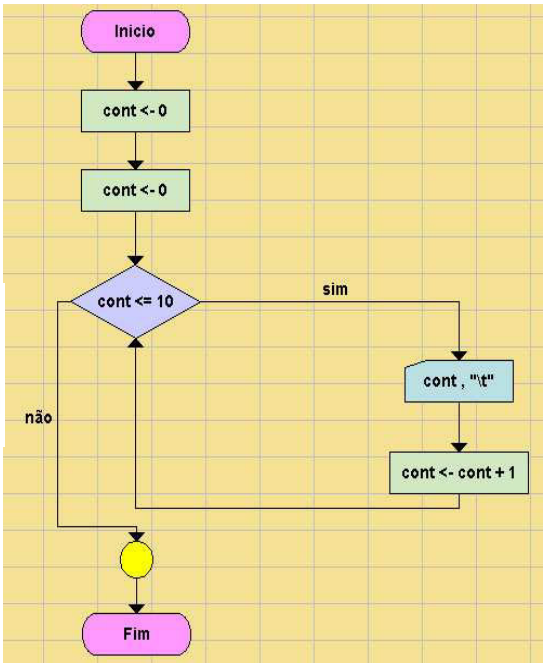
Portugol

```
inicio
    variavel inteiro cont
    para cont de 0 ate 10 passo 1
        escrever cont , "\t"
    proximo
fim
```

Em Java

```
16      public static void main(String[] args) {
17
18          for (int a=1;a<=10;a++)
19      JOptionPane.showMessageDialog(null, "Nemero "+a);
20          }
```

- **Fluxograma**



A estrutura **para** sempre está acompanhado de uma variável contadora que armazena quantas vezes o bloco de sentenças deve ser executado.

Observamos que o programa faz a inicialização, que atribui o valor inicial à variável contadora. Em seguida avalia a expressão, que determina se o valor da variável contadora está dentro do limite desejado.

Caso positivo, o bloco de sentenças é executado e, em seguida, é executada a atualização, que altera o valor da variável contadora. O processo se repete avaliando novamente a expressão.

Exercício Prático

Os Exercícios abaixo devem ser desenvolvidos usando o pseudocódigo e a linguagem Java.

1 - Faça um algoritmo que determine o maior entre **N** números. A condição de parada é a entrada de um valor 0, ou seja, o algoritmo deve ficar calculando o maior até que a entrada seja igual a 0 (ZERO).

2 - Escreva um algoritmo para ler 2 valores e se o segundo valor informado for ZERO, deve ser lido um novo valor, ou seja, para o segundo valor não pode ser aceito o valor zero e imprimir o resultado da divisão do primeiro valor lido pelo segundo valor. (utilizar a estrutura REPITA).

3 - Reescreva o exercício anterior utilizando a estrutura ENQUANTO.

4 - Acrescentar uma mensagem de 'VALOR INVÁLIDO' no exercício [3] caso o segundo valor seja ZERO.

5 - Ler um valor N e imprimir todos os valores inteiros entre 1 (inclusive) e N (inclusive). Considere que o N será sempre maior que ZERO. (***usar a estrutura PARA***)

6 - Modifique o exercício anterior para aceitar somente valores maiores que 0 para N. Caso o valor informado (para N) não seja maior que 0, deverá ser lido um novo valor para N.

Aula 11 – Estruturas de Dados Estáticas:

Estruturas de dados são modelos de armazenamento de dados na memória que visam tornar eficiente tanto o armazenamento quanto o tempo gasto no processamento dos dados. A forma mais simples de uma estrutura de dados é denominada Vetor.

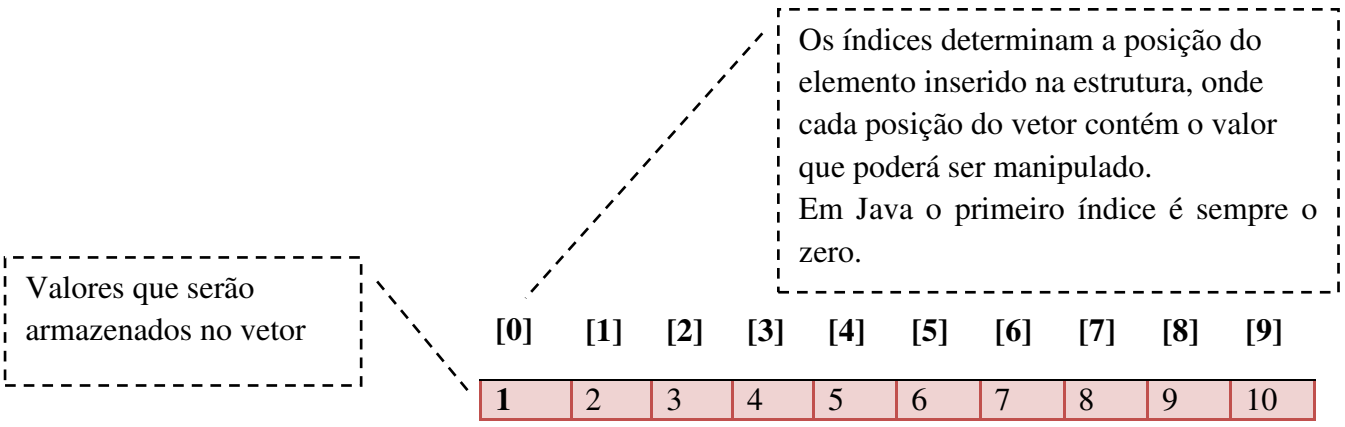
Estruturas de dados estáticas são mais simples e fáceis de usar. Mas para a maioria das aplicações não se pode conhecer o número de elementos necessários na resolução do problema, exigindo estruturas de dados dinâmicas, em que a quantidade de elementos pode ser frequentemente alterada.Em relação ao uso de estruturas de dados dinâmicas podem ser citada, por exemplo; lista de contatos de uma agenda, lista de clientes de um banco, número de personagens de um jogo, lista de usuários de um computador.

11.1 - Vetores - (Array)

Vetores são elementos que são armazenados de forma indexada. Os vetores podem ser estruturas de dados estáticas, pois o número de elementos nessa estrutura é determinado no momento de sua alocação na memória; normalmente quando as variáveis são declaradas. (Algumas linguagens permitem vetores de alocação dinâmica).

O vetor é uma coleção de variáveis de um mesmo tipo de dados, que compartilham o mesmo nome porem com posições diferentes em memória.

Abaixo temo a representação de um vetor de 10 elementos.



11.1.1 – Declaração de vetor

Para declararmos um vetor é necessário definirmos o seu nome, tipo e tamanho, onde o nome identifica o vetor, o seu tipo de dados dos elementos do vetor e o tamanho determina quantos valores o vetor poderá armazenar.

Sintaxe em Portugol

<Tipo de dados> <nome do vetor> [tamanho do vetor]
variavel inteiro vet [6]
Neste exemplo temos um vetor inteiro com o nome de vet com 6 posições.

Sintaxe no pseudocódigo

V : vetor [0..N] de inteiros
Neste exemplo V é o nome do vetor que armazena um conjunto de números inteiros que serão identificados como V[0], V[1], V[2], ..., V[N],

Sintaxe em Java

<Tipo de dados> <identificador> [ ] ou <Tipo de dados> [ ]<identificador>
Em Java os vetores são objetos, neste caso alem de declarar temos que cria-lo em memória.

11.1.2 - Acesso e Atribuição de Vetores

```
inicio
      variavel texto meses [12]
variavel inteiro i
meses[0]<-"Janeiro"
meses[1]<-"Fevereiro"
meses [2]<-"março"
meses [3]<-"abril"
meses [4]<-"maio"
meses [5]<-"junho"
meses [6]<-"julho"
meses [7]<-"agosto"
meses [8]<-"setembro"
meses [9]<-"outubro"
meses [10]<-"novembro"
meses [11]<-"dezembro"

para i de 0 ate 11 passo 1
escrever meses[i],"\n"
proximo

fim
```

Uma vez criado um vetor, a atribuição de valores é processada de elemento em elemento, alterando-se o valor do índice do vetor.

Exemplo: Um vetor usado para guardar os nomes dos meses do ano.
Meses : vetor [1..12] de inteiros
Meses [1] ¬ “Janeiro”
Meses [2] ¬ “Fevereiro”
...
Meses [11] ¬ “Novembro”
Meses [12] ¬ “Dezembro”

Exemplo no Portugol IDE

Neste exemplo no Portugol IDE, precisamos utilizar duas variáveis, uma do tipo texto para armazenar os nomes dos meses que é o nosso vetor que irá armazenar 12 posições. A outra é do tipo inteiro para controlar o índice do vetor que como nos já sabemos irá iniciar em 0 até 11, o que nos da 12 posições conforme declarado no vetor.

Para armazenar o valor em cada índice declaramos desta forma, **meses** (que o nome do vetor) **[0]** (dentro do colchete temos a posição do vetor) onde será armazenado o conteúdo **"Janeiro".**

Na estrutura de repetição PARA vai repetir a instrução escrever com o valor de cada índice do vetor, **para i de 0 ate 11 passo 1,** será repetido da posição 0 até 11 a cada um passo.

Para imprimir uma posição especifica do vetor você deve colocar o nome do vetor **meses [3]** (Dentro do colchete você coloca a posição do índice do vetor que será exibido) ficando assim **escrever meses[3].**

Exemplo em Java

```
15      public static void main(String[] args) {
16
17          String meses[] = new String[12];
18          meses[0] = "janeiro";
19          meses[1] = "fevereiro";
20          meses[2] = "março";
21          meses[3] = "abril";
22          meses[4] = "maio";
23          meses[5] = "junho";
24          meses[6] = "julho";
25          meses[7] = "agosto";
26          meses[8] = "setembro";
27          meses[9] = "outubro";
28          meses[10] = "novembro";
29          meses[11] = "dezembro";
30
31          JOptionPane.showMessageDialog(null, meses);
32      }
33  }
```

Neste exemplo em Java, declaramos um vetor do tipo String chamado meses com 12 posições linha 17. Agora precisamos inicializar o vetor com os meses, fazemos isso através dos índices como vemos nas linhas 18 a 29, para cada posição do vetor atribuímos um valor do tipo String.

Na linha 31 usamos o JOptionPane para exibir o conteúdo do vetor, que irá apresentar uma caixa de dialogo com todos os meses atribuídos, para imprimi-los na console precisaremos usar uma estrutura de repetição o **for,** como veremos abaixo.

```
35          for(int i=0;i<meses.length;i++){
36          System.out.println(meses[i]);
37          }
```

Na estrutura de repetição **for** vai repetir a instrução **System.out.println(meses[i]);** onde será exibido o valor de cada índice do vetor, onde **i=0** que é o valor inicial, a condição **i<meses.length** que será a condição de repetição, o length é para o vetor usar o seu próprio tamanho, ao invés de determinar, se determinado um valor maior que o tamanho do vetor isso causará um erro de execução, pois o valor esta fora dos limites do vetor.

Para imprimir uma posição especifica do vetor você deve colocar o nome do vetor **meses [2]** (Dentro do colchete você coloca a posição do índice do vetor que será exibido) ficando assim **System.out.println(meses[2]);**

Abaixo vemos outra forma de declarar e inicializar um vetor
int meses[] = {1,2,3,4,5,6,7,8,9,10,11,12}; o que esta dentro das { } será o conteúdo do vetor, correspondente a cada índice.

Exemplo2: Realizando operações com vetores.

Em Portugol

```
inicio
    variavel inteiro vet [ 6 ] , i , soma , cont
    real media
    para i de 0 ate 5 passo 1
        escrever "Entre com um numero:"
        ler vet [ i ]
    proximo
    escrever "\n\nVoce digitou:\n"
    para i de 0 ate 5 passo 1
        escrever vet [ i ] , " "
    proximo
    escrever "\n\nDe tras para frente:\n"
    para i de 5 ate 0 passo - 1
        escrever vet [ i ] , " "
    proximo
    escrever "\n\nCalculando a soma e a media aritmetica:\n"
    soma <- 0
    para i de 0 ate 5 passo 1
        soma <- soma + vet [ i ]
    proximo
    media <- soma / 6.0
    escrever "A soma vale " , soma , " e a media vale " , media
    escrever "\n\nQuantos numeros estao acima da media?\n"
    cont <- 0
    para i de 0 ate 5 passo 1
        se vet [ i ] > media entao
            cont <- cont + 1
        fimse
    proximo
    escrever cont , " estao acima da media"
fim
```

Neste exemplo, solicita ao usuário entrar com 6 números que serão armazenados em um vetor, depois irá exibir o conteúdo do vetor, os números de traz para frente, realizará a soma de todos os valores lidos depois calcular a media aritmética e exibir o valor da mesma.

Em Java

```
    public static void main(String[] args) {
51          double media;
52          int cont = 0,soma=0,i;
53          String num="",numinverso="";
54          Integer vet[]=new Integer[6];
55
56          for(i=0;i<vet.length;++i){
57          num=JOptionPane.showInputDialog("Digite um numero: "+i);
58          vet[i]=Integer.parseInt(num);
59                }
60          JOptionPane.showMessageDialog(null, "Você Digitou:");
61  JOptionPane.showMessageDialog(null, vet);
62  for(i=vet.length-1;i>=0;i--){
63  numinverso+=vet[i];
64  }
65  JOptionPane.showMessageDialog(null,"De tras para frente:");
66  JOptionPane.showMessageDialog(null, numinverso);
67  JOptionPane.showMessageDialog(null, "Calcculando a soma e a media");
68  for(i=0;i<vet.length;++i){
69    soma=soma+vet[i];
70  }
71  media=soma/vet.length;
72   JOptionPane.showMessageDialog(null,"A soma vale " + soma + " e a media vale " + media );
73      JOptionPane.showMessageDialog( null,"\n\nQuantos numeros estao acima da media?\n");
74  for(i=0;i<vet.length;++i){
75  if (vet[i]>media)
76      cont=cont+1;
77  }
78      JOptionPane.showMessageDialog(null,"estao acima da media: "+cont);
79      }
80  }
```

11.2 - Matrizes

Estruturas indexadas que necessitam de mais que um índice para identificar um de seus elementos;

São chamadas matrizes de dimensão **n**, onde **n** representa o número de índices requeridos; Uma matriz de dimensão 2 é uma matriz que exige dois índices para identificar um elemento em sua estrutura;

A maioria das linguagens não impõe limite sobre a dimensão de uma estrutura indexada, ficando a cargo de o programador utilizar tantos índices quanto considerar convenientes.

11.2.1 - Declaração

A declaração de uma matriz poderia ser feita da seguinte forma:

Sintaxe Pseudocódigo

Var Vendas : vetor [1..n,1..n] de inteiros

Sintaxe Portugol

<Tipo de dados> <nome do vetor> [tamanho matriz (linha)] [tamanho matriz (coluna)]

Essa declaração é muito semelhante à declaração de vetor, porque o vetor é uma matriz de dimensão 1;

A convenção mais comum é dizermos que o primeiro índice identifica uma linha de uma matriz bidimensional e o segundo, uma coluna;

Sintaxe em Java

<Tipo de dados> <identificador> [ ] [ ]

int tabela [ ] [ ]=new int[3][3]; Exemplo de declaração e inicialização de uma matriz.

Em Java não tem suporte para vetores (arrays) multidimensionais como existe em outras linguagens, para obtermos esta funcionalidade multidimensional devemos declarar um array de array, como mostrado na sintaxe acima.

11.2.2 - Acesso e Atribuição de Matrizes

Para realizarmos o acesso e atribuição em uma matriz temos que lembrar que, cada elemento de uma matriz pode armazenar um valor, para realizar uma atribuição temos que informar as dimensões (linha,coluna), podemos compara com uma planilha que aprendemos a usar em Informática básica.

Vejamos exemplos de algoritmos em Portugol e em Java usando matrizes.

Exemplo: Desejamos guardar uma quantidade de coisas em nosso caixote, o formato dele é uma matriz de 2x2, em cada espaço temos que armazenar um valor de um objeto qualquer, abaixo temos uma tabela com os valores que serão inseridos.

12	**26**
21	**45**

Como já vimos à atribuição em uma matriz é igual como é feito em um vetor, a diferença é que você precisa indicar as coordenadas assim como estudamos no plano cartesiano.

```
inicio
      variavel inteiro tabela [2][2]
tabela[0][0]<-12
tabela[1][0]<-21
tabela[0][1]<-26
tabela[1][1]<-45

escrever tabela [0][0]," "
escrever tabela[0][1],"\n"
escrever tabela[1][0]," "
escrever tabela [1][1]," "
fim
```

Neste exemplo declaramos uma matriz de 2x2 e atribuímos os valores conforme a tabela informada anteriormente, dando a localização de linha e coluna (tabela [0] [0]), onde tabela é o nome da minha matriz de inteiros, se quero armazenar o valor 26 conforme a tabela anterior, então passo linha 0 e coluna 1, para realizar a alocação conforme a tabela.

Exemplo em Java

```
37      public static void main(String[] args) {
38
39          int tabela [] []=new int[2][2];
40
41  tabela[0][0]=12;
42  tabela[1][0]=21;
43  tabela[0][1]=26;
44  tabela[1][1]=45;
45  System.out.print(tabela [0][0]+" ");
46  System.out.print( tabela[0][1]+"\n");
47  System.out.print(tabela[1][0]+" ");
48  System.out.println(tabela [1][1]+" ");
        }
```

Neste exemplo para obter o mesmo resultado da tabela usamos o print que imprime o resultado na mesma linha.

Exercício Prático

1 - Faça um programa em Java e em Portugol que armazene uma lista de contatos com 10 posições.

2 - Escreva um algoritmo no Portugol que receba 10 números e exiba a ordem inversa desses números lidos, depois transcreva para linguagem Java.

3 - Escreva em pseudocódigo e um programa em Java, uma matriz de 4x4 que será preenchida pelo usuário e exibida o resultado na tela.

4 - Vamos desenvolver um programa que receba os valores de uma matriz 2x2, calcule e mostre a matriz resultante da multiplicação dos elementos da matriz 2x2 pelo seu maior elemento.

Fase 3 – Procedimentos e Funções

Aula 12 – Procedimentos

Procedimentos são rotinas (trechos ou módulos) de programas, capazes de executar uma tarefa definida pelo programador. Os programas desenvolvidos com procedimentos são ditos 'modulares'. Os programas desenvolvidos com procedimentos são mais legíveis e melhor estruturados.

Todo procedimento deverá ter um nome e um corpo (conjunto de instruções) e deverá ser escrito no campo de declaração de variáveis, imediatamente abaixo destas. Sua sintaxe pode ser vista abaixo:

Sintaxe em pseudocódigo

```
Procedimento Soma
variável v1,v2,soma: real
Inicio
Ler v1,v2
Soma<-v1+v2
Escrever soma
fim
```

Sintaxe em Java

```
Static void nome_método(argumentos){
Instruções
}
```

Em Java os módulos, sejam procedimentos ou funções, são representados por métodos, nestas aulas vamos construir o nosso conhecimento base sobre métodos, na disciplina de Programação orientada a objetos, será aprofundado.

Todo procedimento possui um espaço para declaração de variáveis, chamadas de variáveis locais, embora não seja obrigatório o seu uso. As variáveis declaradas no programa principal são chamadas de variáveis globais.

Dentro do procedimento podem ser utilizadas tanto variáveis locais quanto variáveis globais. Todas as variáveis locais aos procedimentos são alocadas somente quando o procedimento entra em execução, mas são liberadas quando o procedimento termina, perdendo assim, seus conteúdos.

Caso seja necessário o aproveitamento dos dados manipulados, o procedimento deverá utilizar as variáveis globais.

Para executar o procedimento (comandos), o mesmo deverá ser acionado pelo nome. Para exemplificarmos, vamos imaginar um procedimento chamado **Cálculo**, que serve para calcular uma operação matemática qualquer sobre as variáveis **n1** e **n2**. Então, poderíamos ter o seguinte trecho do programa:

Em Java

```
43    static void somar(){
44      int n1,n2,result;
45          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
46            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
47           result= n1+n2;
48           JOptionPane.showMessageDialog(null, result);
49      }
50
51  public static void main (String []args){
52  somar();
53
54  }
```

Neste exemplo temos um módulo (método) chamado somar, dentro do mesmo temos uma operação de calculo que já estamos acostumados a fazer, realiza a leitura das variáveis, operação de soma e exibe o resultado em uma caixa de dialogo.

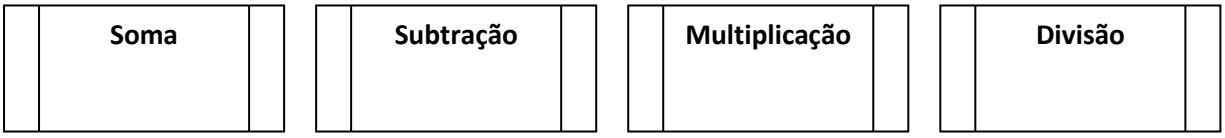
Observe que o método foi criado fora do método main, dentro do mesmo temos apenas a chamada do método que será executado, dentro do main eu posso fazer a chamada de vários métodos (procedimentos ou funções).

Nesta fase vamos focar mais no desenvolvimento de módulos (métodos) em Java, para nos prepararmos uma base de conhecimento para as próximas etapas.

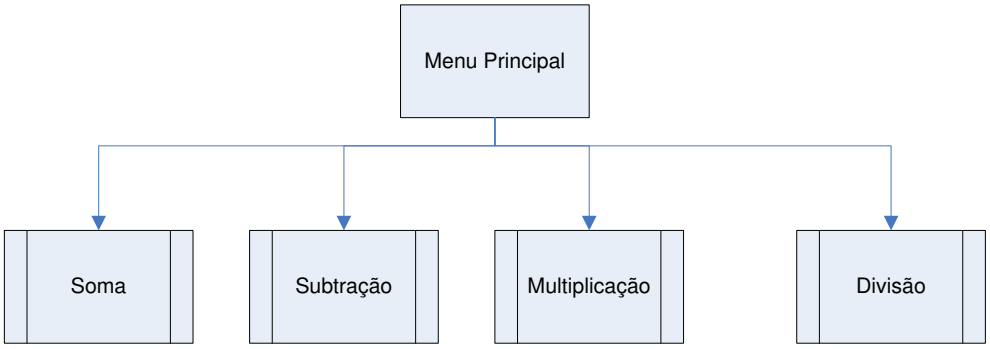
Fluxograma

Vamos utilizar o símbolo abaixo para representar os procedimentos, que são processos definidos.

Processos são ações que podem ser executadas quando solicitadas, abaixo temos 4 módulos ou procedimentos, estes procedimentos dentro deles são utilizados os mesmos símbolos de leitura e exibição de dados que aprendemos no início dos fluxogramas, este símbolo é chamado processo definido.



Abaixo temos um algoritmo representado através de fluxograma que realiza operações aritméticas.



Vamos ver como seria este programa em Java, vamos utilizar a estrutura de seleção caso no menu principal.

```
14  public class OperacoesFunc {
15
16      static void dividir(){
17   int n1,n2,result;
18          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
19            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
20           result= n1/n2;
21           JOptionPane.showMessageDialog(null, result);
22      }
23      static void subtrair(){
24   int n1,n2,result;
25          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
26            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
27           result= n1-n2;
28           JOptionPane.showMessageDialog(null, result);
29      }
30
31      static void multiplicar(){
32      int n1,n2,result;
33          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
34            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
35           result= n1*n2;
36           JOptionPane.showMessageDialog(null, result);
37      }
38
39     static void somar(){
40        int n1,n2,result;
41          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
42            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
43           result= n1+n2;
44           JOptionPane.showMessageDialog(null, result);
45      }
```

Na imagem acima temos os métodos com os procedimentos, abaixo vamos fazer a chamada dos procedimentos.

```
47  public static void main (String []args){
48  int op=0;
49  JOptionPane.showMessageDialog(null, "Escolha uma das opções pra realizar uma operação");
50  JOptionPane.showMessageDialog(null, "1-Soma | 2-Subtrair | 3-Multiplicar | 4-Dividir");
51  op=Integer.parseInt(JOptionPane.showInputDialog("Digite uma das opções anteriores"));
52  switch (op)
53          {
54              case 1: somar();
55              break;
56              case 2: subtrair();
57              break;
58              case 3: multiplicar();
59              break;
60              case 4: dividir();
61              break;
62              default: System.out.println("Opção Errada");
63  }
64      }
65  }
```

Observe que nesta estrutura onde fazemos a chamada dos procedimentos, após escolher uma opção ele executa e depois que realiza a operação o programa é encerrado, para isso não acontecer o que poderíamos usar? *R: Uma estrutura de repetição, por exemplo, o enquanto.*

Aula 13 – Escopo de variáveis

Já sabemos que as variáveis devem ser declaradas antes de serem usadas no programa.
Existem duas formas de declaração de variáveis.

As variáveis locais onde são declaradas dentro de um procedimento ou função. Estas variáveis são utilizadas dentro do bloco de comandos, estas não são acessíveis por outras partes do programa.

As variáveis globais devem ser declaradas em um local que possa ser conhecida por todos os componentes do programa, normalmente no início do programa após o nome. Uma variável global pode ser usada por qualquer componente do programa.

Por exemplo, no programa anterior temos as variáveis n1,n2 e result que realizam operações diferentes têm acesso às mesmas variáveis, porem em cada procedimento repetimos a sua declaração se fossem variáveis globais estariam declaradas uma única vez depois do nome do programa e poderiam ser utilizadas por todos os procedimentos.
Vejamos o exemplo de como ficariam estas variáveis.

```
14  public class OperacoesFunc2 {
15
16      static  int n1,n2,result;
17
18      static void dividir(){
19
20          n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite o primeiro numero"));
21            n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite o segundo numero"));
22          result= n1/n2;
23          JOptionPane.showMessageDialog(null, result);
24      }
```

Agora vamos melhorar o nosso programa, criando as variáveis globais.
Mais Por que usar variáveis globais?

R: Para economizarmos memória e reutilizar as declarações das mesmas.

Aula 14 – Funções

Funções são rotinas similares aos procedimentos, só que retornam um valor após cada chamada. Uma função não deverá simplesmente ser chamada, como no caso dos procedimentos, mas deverá ser atribuída a alguma Var. Uma função deve ter um nome e um tipo, e sua sintaxe é mostrada abaixo.

Static <tipo de dados> <nome do método>(argumentos){
Instruções
return <nome da variável de retorno>
}

O nome da função é quem deve assumir o valor da função, como se fosse uma variável. Assim como nos procedimentos, uma função deve ser definida dentro do espaço de declaração de variáveis do programa. Para que a função possa retornar um valor, este deverá ser explicitamente atribuído ao nome da função, dentro da rotina da função.

Vejamos uma função para realizar uma soma de 2 números.

```
14  public class Funcoes {
15
16      static int   result;
17      static int  somar(int n1,int n2){
18
19              result= n1+n2;
20
21          return result;
22      }
```

Declaramos a função e colocamos a operação que ela deve fazer, e vamos retornar a variável result, que é a soma de n1+n2 linhas 17 a 21.

Aula 15 – Parâmetros

Os parâmetros são valores que são passados de um algoritmo principal para os módulos, no exemplo abaixo o que estamos fazendo é a passagem de parâmetro, onde do método main passamos os valores digitados pelo usuário e retornamos o valor da soma.

Para chamarmos uma função ou procedimento deste tipo, devemos colocar o nome seguido de um conjunto de parâmetros entre parênteses. Os parâmetros devem ser do mesmo tipo de dados e quantidade, conforme foi definida a função ou o procedimento, abaixo um exemplo de passagem de parâmetro.

```
14  public class Funcoes {
15
16      static int   result;
17      static int  somar(int n1,int n2){
18
19              result= n1+n2;
20
21          return result;
22      }
23
24      public static void main(String[] args) {
25          int v1,v2,resultado;
26          v1=Integer.parseInt(JOptionPane.showInputDialog("Digite o 1º valor"));
27          v2=Integer.parseInt(JOptionPane.showInputDialog("Digite o 2º valor"));
28          resultado=somar(v1, v2);
29          JOptionPane.showMessageDialog(null, result);
30      }
31
32  }
```

No método main, declaramos as variáveis locais que irão receber os valores do usuário, após isso dizemos que resultado=(recebe)a função(ou método) somar onde passo os 2 valores obtidos do usuário, o método realiza a operação no caso soma e retorna o valor da soma dos 2 números passados.

Agora podemos melhorar o nosso programa que realiza as operações aritméticas, vamos ao trabalho, iremos colocar em nosso programa agora todos os conceitos aprendidos até agora.

Aula 16 – Vamos a prática

Esta aula é dedicada a pratica e reflexão do que estamos aprendendo, vamos unir todo o conhecimento adquiridos para resolver os problemas, para os exercícios abaixo iremos aplicar os conceitos da fase.

1 - Escreva um programa para calcular a soma dos números positivos, inferiores ou iguais a 1000.

2 - Escreva um programa para calcular o fatorial de um número.

3 - Escreva um programa para mudar uma lâmpada fundida de um candeeiro.

4 - Escreva um programa para calcular o maior de três números inteiros.

5 - Escreva um programa para calcular o maior e o menor de três números inteiros.

6 - Escreva um programa para calcular, dados três números inteiros, a soma dos dois maiores.

7 - Escreva um programa para calcular o máximo divisor comum de dois números inteiros.

8 - Escreva um programa para calcular o mínimo múltiplo comum de dois números inteiros.

Fase 4 – Resolução de problemas

Esta fase é dedicada a trabalhar a capacidade de resolução de problemas escolar ou pessoal, e aprimorar a capacidade de raciocínio rápido diante de situações diversas.

A prática nos proporciona a solução direta e eficaz para a solução de um problema, acabaremos aplicando essa solução rotineiramente, e a tarefa servirá, simplesmente, para exercitar habilidades já adquiridas'. (POZO e ECHEVFRRIA, 1998, p. 1 7).

A resolução de problemas tem grande poder motivador para o aluno, pois envolvem situações novas e diferentes atitudes e conhecimentos.

Para que seja possível a resolução de um problema são necessárias várias habilidades:
Em POZO e ECHEVERRÍA (1998) encontram-se os "passos necessários para resolver um problema, segundo Poya." (quadro 1)

Quadro 1

Habilidades para resolução de um problema
Qual é a incógnita?
Compreender o problema
Quais são os dados?
Qual é a condição?
A condição é suficiente para determinar à incógnita?
É suficiente'? Redundante? Contraditória?
Conceber um plano Já encontrou um problema semelhante?
Já viu o mesmo problema proposto de maneira um pouco diferente?
Conhece um problema relacionado com este?
Conhece algum teorema que possa lhe ser útil?
Olhe a incógnita com atenção e tente lembrar um problema que lhe seja familiar ou que tenha a mesma incógnita, ou uma incógnita similar.

Para a compreensão de um problema não é suficiente compreender as palavras ou a linguagem e os símbolos apresentados, mas é necessário assumir a busca da sua solução; superando dificuldades e obstáculos apresentados.

Após compreendermos um problema, temos que elaborar um plano de ação que permita a sua resolução, quais os procedimentos que deverão ser utilizados para que seja alcançada a meta final, depois seguirmos o passo a passo da execução do plano elaborado.

E por ultimo fazermos o retrospecto, revendo todo o caminho percorrido para se chegar à solução, podendo auxiliar na determinação e correção de erros eventuais.

Aula 17 - Problemas matemáticos

Nesta aula temos que resolver os problemas abaixo usando todos os conhecimentos adquiridos, para gerarmos o melhor algoritmo da solução será implementado em uma linguagem de programação Java.

1 - Um fazendeiro colheu milho, feijão e café, num total de 14.400 sacas, sendo que a quantidade de sacas de café foi o quádruplo da quantidade de sacas de feijão. Se tivesse colhido mais 2.400 sacas de feijão, o total dessas sacas atingiria o total de sacas de milho. Faça um programa em Java para saber o total de sacas de café.

2 - Em nossa cidade, o preço pago por uma corrida de táxi inclui uma parcela fixa, denominada bandeirada, e uma parcela que depende da distância percorrida. A bandeirada custa R$ 4,20 e cada quilômetro rodado custa R$ 0,65.

a) Faça um programa que receba a quilometragem rodada e informe quanto foi à corrida deste taxista.

b) Um determinado passageiro pagou R$ 14,20 em uma corrida, agora ele quer saber quantos quilômetros ele viajou, vamos ajuda-lo fazendo um programa que descubra quantos quilômetros ele viajou para pagar o valor acima.

3 - João e Paulo receberam juntos, R$ 630,00 para consertar uns computadores. Se João gastar R$ 60,00 do que recebeu, e Paulo gastar R$ 90,00, ambos ficarão com quantias iguais. Qual foi a quantia recebida por Paulo? Faça um programa para descobrir quanto cada um recebeu.

4 - Dentre os 1.250 médicos que participaram de um congresso, 48% eram mulheres. Dentre as mulheres 9% eram pediatras. Faça um programa em Java que possa responder as perguntas abaixo;

a) Quantas mulheres pediatras participaram desse congresso?

b) Quantas mulheres participaram desse congresso?

c) Quantos homens participaram desse congresso?

5 - De uma pesquisa com alunos egressos, em que foram entrevistados 625 estudantes do curso de Técnico em Informática, concluiu-se que 84 deles trabalham. Dos estudantes entrevistados, quantos trabalham? Faça um programa para descobrir quantos trabalham.

6 - Para a confecção de uma peça metálica, foram fundidos 15 kg de cobre, 9,75 kg de zinco e 0,25 kg estanho. Qual é a porcentagem de cobre dessa peça? Faça um programa que receba os valores de cobre, zinco e estanho e mostre a porcentagem de cobre da peça.

7 - Na compra de uma camisa tive um desconto de 12% sobre o preço de etiqueta. Qual era o preço de etiqueta, sabendo que o desconto foi de R$ 2,40? Faça um programa em Java que mostre o preço da etiqueta.

8 - Escreva um algoritmo para ler a o número total de eleitores de um município, o numero de votos brancos, nulos e válidos. Calcular e escrever o percentual que cada um representa em relação ao total de eleitores.

9 - Uma revendedora de carros usados paga a seus funcionários vendedores um salário fixo por mês, mas uma comissão também fixa para cada carro vendido e mais 5% do valor das vendas por ele efetuadas. Escreva um algoritmo que leia o numero de carros por ele vendidos, o valor total de suas vendas, o salário fixo e o valor que ele receber por carro vendido. Calcule e escreva o salário final do vendedor.

10 - Escreva um algoritmo para ler uma temperatura em graus Fahrenheit, calcular e escrever o valor correspondente em graus Celsius (baseado na formula abaixo):

C/5 = F-32/9

Obs.: Saiba que 100 Graus Celsius correspondem a 212F.

a) após a solução implementada, faça um programa que receba 10 temperaturas realize a conversão e mostre o resultado.

11 - Faça um algoritmo que leia três notas de um aluno, calcule e escreva a media final deste ano. Considerar que a media final é ponderada e que o peso das notas é 2, 3 e 5. Formula para calculo da media final é:

M=n1*n2*n3+n3*5/10

12 - Ler dois valores (*não permitir que tenham valores iguais*) e escreva o maior deles.

13 - Ler um valor e escrever se é positivo, negativo ou zero.

14 - Ler 3 valores (*considere que não serão informados valores iguais*) e escrever o maior deles.

15 - Escreva um algoritmo para ler as notas da 1ª e 2ª avaliações de um aluno, calcule e imprima à média (simples) desse aluno. Só devem ser aceitos valores válidos durante a leitura (0 a 10) para cada nota.

17.1 - Estudos de caso

SISTEMA DE TELEMARKETING

Hoje em dia uma das formas de venda mais utilizada é o telemarketing, que obtêm uma lista de possíveis compradores para os produtos vendidos e seus respectivos telefones e utiliza um time de vendedores para ligar para esse conjunto de pessoas.

José Augusto é o dono da JA Telemarketing. Sua empresa realiza venda dos produtos mais variados para diversas companhias.

Ele possui uma equipe de *N* vendedores e uma lista de ligações a serem feitas. Para cada ligação sabe-se o tempo *T* em minutos que ele vai durar. Os vendedores são identificados por números de um a *N* e fazem as ligações da seguinte forma:

- Inicialmente, todos os vendedores são inativos.
- Sempre que um vendedor realiza uma ligação ele ficará ocupado por *T* minutos descritos na lista para aquela ligação. O tempo entre duas ligações consecutivas do mesmo vendedor é desprezível.
- Um vendedor não pode fazer mais de uma ligação ao mesmo tempo.
- Um vendedor que esteja inativo deverá fazer a ligação que estiver no topo da lista. Caso mais de um vendedor esteja inativo no mesmo instante, o vendedor com menor identificador dentre os vendedores inativos deverá fazer a ligação que esta no topo da lista.
- Assim que uma ligação é atribuída a um vendedor, ela é removida da lista.
- Um vendedor fica inativo sempre que termina uma ligação.

Por exemplo, suponha que um time de 4 vendedores deve fazer 6 ligações , cujos tempos sejam 5,2,3,3,4,9. Como inicialmente nenhum vendedor esta ocupado, o primeiro vendedor fará a ligação de 5 minutos, o segundo vendedor a ligação de 2 minutos e os vendedores de número 3 e 4 farão ligações de 3 minutos. Como o segundo vendedor terminará a sua ligação antes dos demais, ele fará a quinta ligação, de 4 minutos e, por fim, o terceiro vendedor (cujo tempo é igual ao do quarto vendedor, mas o numero é menor) fará a sexta ligação, de 9 minutos.

Para ajudarmos o nosso empresário e ganharmos um dinheiro extra, devemos desenvolver um programa que, receba o numero de vendedores, o número de ligações e a duração de cada ligação, e depois determinar o numero de ligações feitas por cada vendedor.

Então temos;

- *N* que é o numero de vendedores.
- *L* indicando o numero de ligações e serem realizadas
- *T* representa a duração de cada ligação.

Então ($1 \leq T \leq 1.000$, $1 \leq L \leq 1.000.000$) e ($1 \leq T 30 \leq 30$)

NORDESTE AVIÕES

A Nordeste aviões tem um cliente chamado Sr. Antonio que esta acostumado a fazer viagens de avião, sempre na classe econômica, ele quer saber qual seu assento com base no novo sistema da companhia aérea.

Toda a aeronave contém uma classe executiva e uma econômica, sendo que as primeiras fileiras do avião pertencem a classe executiva e as restantes a classe econômica.

Cada assento do avião é indicado por um número correspondente a sua fileira e por uma letra que indica a sua posição na fileira, sendo **A** a posição mais a esquerda da fileira, B a posição a direita do assento A, C o assento a direita do assento B, e assim por diante, seguindo o alfabeto de 26 letras. Por exemplo, o assento 9B está localizado na nona fileira, logo a direita do assento 9A.

Na figura abaixo podemos ver a numeração utilizada em uma aeronave com as nove fileiras de três assentos cada.

1A	1B	1C
2A	2B	2C
3A	3B	3C
4A	4B	4C
5A	5B	5C
6A	6B	6C
7A	7B	7C
8A	8B	8C
9A	9B	9C

A companhia aérea adotou, para a classe econômica, um sistema no qual o bilhete indica a posição do passageiro na fila de embarque e não seu assento no voo. A fila de embarque contem apenas passageiros da classe econômica.

Sr. Antonio descobriu que o primeiro passageiro da fila de embarque deve sempre sentar-se no assento localizado na primeira fileira da classe econômica, posição A. O segundo passageiro deve sentar-se nesta mesma fileira, posição B, e assim por diante, até que todos os assentos dessa fileira estejam ocupados.

Esse processo é repetido a cada fileira da classe econômica, ate que acabem os assentos desta classe ou todos os passageiros da fila já tenham embarcado.

Caso a classe econômica já esteja lotada e ainda haja passageiros na fila, esses passageiros embarcarão somente no próximo voo.

Como viajante frequente, Sr. Antonio conhece bem os diversos modelos de aeronaves e é capaz de dizer o numero total de fileiras no avião, o numero de posições por fileira, e a partir de que fileira começa a classe econômica.

Com base nessas informações, ele pediu a sua ajuda para descobrir, a partir de sua posição na fila, se ele tem assento garantido neste voo e, caso tenha, qual seu assento.

Então temos:

- F: Numero total de fileiras no avião.
- C: Numero de posições por fileira.
- E: Numero da primeira fileira da classe econômica.
- B: Posição na fila de embarque do Sr. Antonio.
  Onde: F, C, E, B ($2 \leq F \leq 1.000$, $1 \leq C \leq 26$, $1 \leq E \leq F$, $2 \leq B \leq 50.000$)

Referências Bibliográficas

Algoritmos e Lógica de Programação – THOMSON – Marcos Antônio Furlan de Souza, Marcelo Marques Gomes, Marcio Vieira Soares, Ricardo Concilio.

DEITEL, Harvey M.; DEITEL, Paul J. Java como Programar 8ª Ed. Editora Pearson.

Fundamentos da Programação de Computadores – Algoritmos, Pascal, C/C++, JAVA 2 Edição – Pearson_Prentice Hall – Ana Fernanda Gomes Ascencio, Edilene Aparecida Veneruchi de Campos.

Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados – São Paulo: Forbellone, André Luiz Villar - MAKRON, 1993.

PUGA, Sandra; RISSETTI Gerson Lógica de Programação e Estruturas de Dados. Editora Pearson.

PUGA, Sandra; RISSETTI Gerson. Lógica de Programação e Estruturas de Dados com aplicações em Java. Editora Pearson.

GOMES, Ana F.; VENERUCHI, Edilene A.C. Fundamentos da programação de computadores 2ª Ed. Editora Pearson.

VILLAR André Luiz, EBERSPACHER Henri F. Lógica de programação a construção de algoritmos e estruturas de dados 3ª Ed. Editora Pearson.

GOMES, Ana F.; VENERUCHI, Edilene A.C. Fundamentos da programação de computadores 2ª Ed. Editora Pearson.

PATERLINI, Roberto Ribeiro. **Fórmula versus algoritmo da resolução de um problema.** RPM n.º 27, 1.º quadrimento de 1995.

ECHEVERRÍA, M.D.P.P.; POZO, J.I. Aprender a resolver problemas e resolver problemas para aprender. In: **A solução de problemas**: aprender a resolver, resolver a aprender. Juan Ignacio Pozo. Porto Alegre: Artmed, 1998.

PÓLYA, G. **A Arte de Resolver Problemas**. Trad. Heitor Lisboa de Araújo. Rio de Janeiro: Editora Interciência, 1978.

SOARES, M. T. C; PINTO N.B. **Metodologia da resolução de Problemas**. 24ª Reunião Anual da ANPED, Caxambu, MG, 2001.

Sites na Web:
http://www.inf.pucrs.br/~egidio/algo1/

http://www.dei.estt.ipt.pt/portugol/

http://grupologica.forumslog.com/

http://olimpiada.ic.unicamp.br/

Softwares

Instituto Politécnico de Tomar
Escola Superior de Tecnologia de Tomar
PORTUGOL 2.3
António Manso
Paulo Santos
Luís Oliveira
Eduardo Santana

Hino Nacional

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da liberdade, em raios fúlgidos,
Brilhou no céu da pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada,Brasil!

Deitado eternamente em berço esplêndido,
Ao som do mar e à luz do céu profundo,
Fulguras, ó Brasil, florão da América,
Iluminado ao sol do Novo Mundo!

Do que a terra, mais garrida,
Teus risonhos, lindos campos têm mais flores;
"Nossos bosques têm mais vida",
"Nossa vida" no teu seio "mais amores."

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, de amor eterno seja símbolo
O lábaro que ostentas estrelado,
E diga o verde-louro dessa flâmula
- "Paz no futuro e glória no passado."

Mas, se ergues da justiça a clava forte,
Verás que um filho teu não foge à luta,
Nem teme, quem te adora, a própria morte.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Hino do Estado do Ceará

Poesia de Thomaz Lopes
Música de Alberto Nepomuceno
Terra do sol, do amor, terra da luz!
Soa o clarim que tua glória conta!
Terra, o teu nome a fama aos céus remonta
Em clarão que seduz!
Nome que brilha esplêndido luzeiro
Nos fulvos braços de ouro do cruzeiro!

Mudem-se em flor as pedras dos caminhos!
Chuvas de prata rolem das estrelas...
E despertando, deslumbrada, ao vê-las
Ressoa a voz dos ninhos...
Há de florar nas rosas e nos cravos
Rubros o sangue ardente dos escravos.
Seja teu verbo a voz do coração,
Verbo de paz e amor do Sul ao Norte!
Ruja teu peito em luta contra a morte,
Acordando a amplidão.
Peito que deu alívio a quem sofria
E foi o sol iluminando o dia!

Tua jangada afoita enfune o pano!
Vento feliz conduza a vela ousada!
Que importa que no seu barco seja um nada
Na vastidão do oceano,
Se à proa vão heróis e marinheiros
E vão no peito corações guerreiros?

Se, nós te amamos, em aventuras e mágoas!
Porque esse chão que embebe a água dos rios
Há de florar em meses, nos estios
E bosques, pelas águas!
Selvas e rios, serras e florestas
Brotem no solo em rumorosas festas!
Abra-se ao vento o teu pendão natal
Sobre as revoltas águas dos teus mares!
E desfraldado diga aos céus e aos mares
A vitória imortal!
Que foi de sangue, em guerras leais e francas,
E foi na paz da cor das hóstias brancas!

GOVERNO DO
ESTADO DO CEARÁ

Secretaria da Educação