UNIVERSIDADE
FUMEC/FACE
Faculdade de Ciências Empresariais

Linguagens e Técnicas de Programação - LTP

Java

CIÊNCIA DA COMPUTAÇÃO

PROF. AIR RABELO

2025

Sumário

1 - Introdução

1.1 - Conceito de Algoritmo

É um conjunto de ordens ou comandos que deverão ser executados em uma seqüência determinada para resolver uma tarefa.

É a descrição seqüencial de ações a serem executados para o cumprimento de uma determinada tarefa.

É a forma pela qual descrevemos soluções para problemas que serão implementadas posteriormente em uma linguagem de programação e executadas pelo computador.

Pode ser escrito através de pseudo linguagens, como o Portugol, ou símbolos como em fluxogramas.

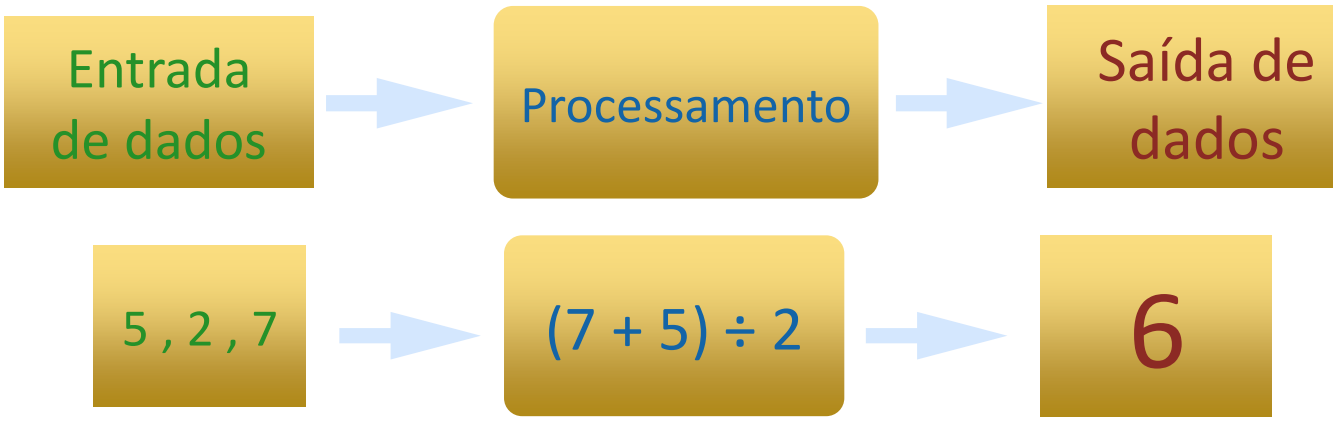
1.2 - Princípio básico de um algoritmo

O algoritmo segue o mesmo princípio básico de operações em um computador:

1 - Entrada de Dados: é processo pelo qual os dados são inseridos no computador.

2 - Processamento dos Dados: é o conjunto de operações / cálculos que são executados com os dados inseridos na Entrada de Dados.

3 - Saída de Dados: é o resultado do processamento dos dados.



1.3 - Exemplo de Algoritmo

Preparando um sanduíche:

Etapa 1 - Pegar os ingredientes necessários: pão, manteiga, queijo, ovo, carne.	Entrada de dados
Etapa 2 - Cortar o pão ao meio Etapa 3 - Passar a manteiga no pão Etapa 4 - Cortar o queijo Etapa 5 - Colocar o queijo no pão Etapa 6 - Grelhar a carne Etapa 7 - Colocar a carne em cima do queijo Etapa 8 - Fritar o ovo Etapa 9 - Colocar o ovo em cima da carne e fechar o pão	Processamento dos dados
Etapa 10 - Entregar o sanduíche pronto para o cliente	Saída dos dados

1.4 - No computador, como os algoritmos são escritos ?

São escritos utilizando uma linguagem de programação como Pascal, linguagem C, Java, e outras, e são chamados de Programas.

1.5 - Algoritmo x Programa

Um programa é a conversão, ou tradução, de um algoritmo para uma determinada linguagem de programação, segundo suas regras de sintaxe e semântica da linguagem, de forma a permitir que o computador possa interpretar e executar a seqüência de comandos pretendidos.

1.6 - Linguagem de Programação

Uma linguagem de programação é um conjunto de símbolos (comandos, identificadores, caracteres ASCII, etc. ...) e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador.

1.7 – Programa Fonte ou Código Fonte

O programa que escrevemos utilizando uma linguagem de programação, como o Pascal, a linguagem C, Java, e outras, é chamado de Programa Fonte. As linguagens de programação permitem que os programas sejam escritos e interpretados pelo homem, mas não pode ser interpretado pelo computador.

1.8 – Programa Objeto ou Código Objeto

Programa Objeto é a seqüência de comandos de um programa escritos em linguagem de máquina. A linguagem de máquina é composta de símbolos binários que é interpretada pelo computador, mas não pode ser interpretada pelo homem.

1.9 – Compilador

Compilador é um programa que traduz o Programa Fonte escrito por um programador em um Programa Objeto a ser interpretado pelo computador. Ou seja, é um tradutor de Programa Fonte em Programa Objeto.

1.10 – A tabela ASCII

ASCII (*American Standard Code for Information Interchange)*, em português: "Código Padrão Americano para o Intercâmbio de Informação". É um conjunto binário de 8 bits (ASCII estendida) que é capaz de representar 256 códigos binários diferentes (2^8 = 256 códigos). Cada sequencia de 8 bits da tabela ASCII (equivalente a um byte) corresponde a um caractere, que pode representar uma letra, número, símbolo, e outros. A tabela foi criada com base no alfabeto inglês / ibérico. Os arquivos criados nos computadores contendo apenas caracteres da tabela ASCII são conhecidos como arquivos texto ou arquivos ASCII. Desenvolvida a partir de 1960, grande parte das codificações de caracteres modernas a herdaram como base.

Exemplos da Tabela ASCII:

Código Binário	Caracter
0100 0001	A
0101 1010	Z
0110 0001	a
0111 1010	z
0100 0000	@
0011 1111	?
0011 0101	5
0011 1001	9

Para escrever um programa fonte devemos utilizar um editor de programa que utilizará somente os caracteres da Tabela ASCII, ou seja, não se utiliza em programas fonte formatações do tipo negrito, itálico, tamanho de fonte, e outra.

O quadro a seguir mostra uma parte da tabela ASCII com os caracteres mais utilizados:

Binário	Decimal	Hexa	Glifo
0010 0000	32	20	
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Decimal	Hexa	Glifo
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Decimal	Hexa	Glifo
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

1.11 – Tipos de Dados

Os dados manipulados ou armazenados na memória do computador são divididos em 3 classes:

1.11.1 – NUMÉRICOS

São valores compostos somente por dígitos e possuem valor de grandeza numérica

Ex: `0 , 1 , 2 , 5.10 , -10 , -2.35 , 1240, ...`

A classe dos valores numéricos é dividida em:

Números inteiros → para representar números inteiros positivos ou negativos

Ex: `5 , -10 , 1250 , 0 , -375 , …`

Números reais → para representar números reais positivos ou negativos

Ex: `11.60 , -10.50 , 11250.357 , ...`

obs: no sistema de numeração da lingua portuguesa é utilizamos no conjunto de números reais a vírgula (,) como o símbolo separador dos valores inteiros e decimais. Entretanto, nas linguagens de programação o símbolo utilizado para separar os valores inteiros dos decimais é o ponto (.)

Exemplos de dados numéricos:

- O aluno possui 1.85 metros de altura;
- O salário do empregado é R$ 3500.25;
- O número da sala de aula é 214;
- A idade do aluno é 26;

1.11.2 – ALFANUMÉRICOS

São valores compostos por um ou mais caracteres da tabela ASCII e são interpretados como um texto.

Ex: `a , bacia , c , Altura , B , Casa , ! , @&( , # , 0 , 10 , X2, 195AB , José Maria , 25/10/2013 ,...`

obs: os algarismos de 0 a 10 estão presentes na tabela ASCII, no entanto, quando um valor Alfanumérico é composto por estes algarismos, este valor não representará uma grandeza quantitativa, ou seja, não possuirá valor numérico. Desta forma, tal valor representará apenas um conjunto de algarismos sem grandeza numérica.

Nas linguagens de programação, para diferenciar os valores alfanuméricos dos numéricos, os valores alfanuméricos são escritos entre aspas ou apóstrofes. Exemplos:

Ex: `"José Maria" , "25/10/2013" , '3' , 'R' , "Relatório de Faturamento", ...`

Exemplos de dados alfanuméricos:

- O título do relatório era: `"Relatório de faturamento"`
- O nome do aluno é: `"José Maria"`
- A data de nascimento do aluno é: `"25/10/2013"`
- A letra que contem a resposta correta é: `'D'`
- O sexo do atleta (M/F) é: `'M'`

1.11.3 - LÓGICOS

Podem assumir apenas dois valores: `verdadeiro` ou `falso`. Usados para representar valores em situações as quais apenas o verdadeiro ou falso são possíveis.

Exemplos de dados lógicos:

- A porta está aberta: `falso`
- A luz está acessa: `verdadeiro`
- Está chovendo agora: `falso`
- O Brasil é um país grande: `verdadeiro`

1.12 – Variáveis de memória

Variável é uma região identificada da memória que tem por finalidade armazenar informações (dados) de um programa em execução. Ou seja, a variável é o nome que damos a um dado armazenado na memória do computador, e todo dado armazenado na memória do computador deve ter um nome que o identifique.

Quando precisarmos armazenar um dado na memória, atribuímos este dado ao nome de uma variável. Exemplo:

→ comando para guardar a camisa na Gaveta1:

```
gaveta1 = "camisa";
```

→ comando para guardar a meia na Gaveta2:

```
gaveta2 = "meia";
```

→ comando para guardar a bermuda na Gaveta3:

```
gaveta3 = "bermuda";
```

Toda vez que precisarmos buscar um dado armazenado na memória utilizamos o nome da variável para identificar o dado. Exemplo:

→ mostrar o que tem na Gaveta1:

```
exibir ( gaveta1 );
```

→ mostrar o que tem na Gaveta2

```
exibir ( gaveta2 );
```

→ mostrar o que tem na Gaveta3

```
exibir ( gaveta3 );
```

Toda variável de memória deve ter um nome e deve estar associada a um tipo de dado.

Toda variável armazena apenas um valor por vez.

Toda vez que um valor é atribuído a uma variável, o seu valor anterior será sobreposto. Portanto as variáveis armazenam apenas o ultimo valor atribuído a elas. Exemplo:

```
idade = 30;
idade = 50;
exibir (idade);
```

Na primeira linha do exemplo acima a variável `idade` recebeu o valor `30`. Naquele momento, o único valor armazenado na variável era `30`.

Na segunda linha a mesma variável `idade` recebeu o valor `50`. Desta forma, o antigo valor `30` foi sobreposto pelo novo valor `50`. Após a execução desta segunda linha, o único valor que a variável `idade` passou a armazenar é o ultimo que ela recebeu, ou seja, `50`.

Na terceira linha, o comando `exibir` irá exibir na tela o conteúdo da variável `idade`. O valor que será exibido na tela será `50`, que é o ultimo valor recebido pela variável `idade`, e por isto é o único valor que estará armazenado nela.

2 - A Linguagem JAVA - Introdução

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa SunMicrosystems.

Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual.

A linguagem Java possui como principais vantagens e/ou características:

- simplicidade na escrita dos programas;	- segurança;
- portabilidade entre diferentes plataformas de sistemas operacionais e hardware;	- orientação a objetos;
- grande similaridade com as linguagens C, C++ e C#;	- recursos de rede;
- vasta biblioteca de classes;	- alta performance;
- entre outras.	

A sintaxe da linguagem Java se assemelha ao C++. Os programas fontes são arquivos do tipo texto gravados com extensão .java . Estes programas fonte devem ser compilados com o copilador javac, para a geração do programa objeto Java com extensão .class , que são interpretados pela Máquina Virtual Java (*Java Virtual Machine*).

Curiosidade: no inglês americano java é uma versão informal da palavra café, ou seja, java significa café, daí o motivo do símbolo da linguagem Java ser uma chícara.

2.1 - Estrutura de um programa em Java:

Todo programa em Java é identificado como uma Classe. A Classe no Java é estruturada da seguinte forma:

```
import NOME_PACOTE_DE_CLASSES;          // nome dos pacotes de classes Java que
                                        // serão utilizados neste programa
public class NOME_CLASSE {          // nome da classe a ser criada

     public static void main (String[] args) {      // método principal

          <bloco de comandos>;
     }
}
```

OBS: O Java faz distinção entre letras minúsculas e letras maiúsculas, portanto, comandos e tipos escritos em letras minúsculas são diferentes de maiúsculas. Alguns comandos são escritos apenas em letras minúsculas, outros em letras maiúsculas, e também podem utilizar maiúsculas e minúsculas.

Os pacotes são arquivos que podem conter várias classes, as classes podem conter vários métodos. A diretiva `import` permite que o programa reconheça as classes do pacote e de seus métodos.

2.2 – Comentários no programa

Comentários são textos inseridos no programa fonte com o objetivo de explicar ou documentar uma lógica, uma variável ou outro objeto qualquer. Os comentários são ignorados pelo compilador, portanto não são levados para o programa objeto. Os comentários podem ocupar uma ou mais linhas no programa, e para inseri-los se utiliza os símbolos `/*...*/` ou `//`

Exemplos:

```
/* comentário 1
comentário 2
comentário 3  */

<comando 1>;   // comentário 1
<comando 2>;   // comentário 2
```

2.3 – Variáveis

Variável é uma região identificada da memória que tem por finalidade armazenar informações (dados) de um programa em execução. Uma variável armazena apenas um valor por vez.

Toda vez que um valor é atribuído a uma variável, o seu valor anterior será sobreposto. Portanto as variáveis armazenam apenas o ultimo valor atribuído a elas.

Sendo consideradocomo valor o conteúdo de uma variável, este valor está associado ao tipo de dado da variável (inteiro, real, lógico, caracter, ...).

2.4 – Nomes de Variáveis e outros

Os nomes criados para as variáveis, constantes, classes, métodos, tipos de dados, e outros objetos devem seguir as seguintes **regras**:

- o primeiro caracter deve ser sempre uma letra, ou caracter *underline (_),* ou o caracter $
- os demais caracteres podem ser **somente:** letras, números, caracter *underline(_)* ou $
- não pode ser utilizada a letra Ç (c com cedilha) e nem letras acentuadas;
- os nomes podem conter qualquer quantidade de caracteres;

Obs: - o Java diferencia letras maiúsculas de minúsculas, portanto, são consideradas diferentes uma da outra. Ex: a variável de nome `Idade_Aluno` é diferente da variável de nome `idade_aluno`

Boas prática para criar os nomes de variáveis:

- os nomes devem ser simples e descritivos, e indicar qual o conteúdo que será armazenado;
- Apesar dos nomes poderem conter qualquer quantidade de caracter, evitar usar nomes muito curtos (porque dificulta indicar qual o conteúdo armazenado) ou muito longos (porque dificultam a escrita e ocupam muito espaço no código fonte o que dificulta a interpretação do código pelo programador).

Nomes segundo os Padrões de codificação Java definidos pela Sun (boas práticas):

- Nomes de variáveis e métodos: devem ser escritos em letras minúsculas, e se forem nomes compostos, para diferenciar uma palavra da outra, a partir da segunda palavra, a primeira letra de cada palavra deve ter letras maiúsculas.

O nome de uma variável deve indicar o que ela armazenará, exemplo:
- variável para armazenar a idade do aluno: `idadeAluno`
- variável para armazenar o peso do atleta: `pesoAtleta`

O nome de um método deve indicar qual a ação ele desempenha, e para isto deverá utilizar um verbo, exemplo:
- método para calcular o salário do empregado: `calcularSalario`
- método para receber a altura do empregado: `receberAltura`

- Nomes de classes: já os nomes de classes devem ser escritos com a primeira letra de cada palavra em maiúsculo e as demais letras em minúsculo, exemplo: `ProgramaExemplo`.

- Nomes de constantes: devem ser escritas com letras maiúsculas, exemplo: `PI` , `VELOCIDADE_LUZ`

2.5 – Grupos de Variáveis

As variáveis criadas por meio de tipos primitivos são divididas em 3 grupos:

- NUMÉRICAS – aceitam somente dígitos, e possuem valor numérico Ex: 0,1,2, 5.10 ...
- ALFANUMÉRICAS – qualquer caracter da tabela ASCII. Ex:a,b,c,A,B,C,!,@,#, 0, 1,2, ...
- LÓGICAS - assumem apenas dois valores: `true` ou `false`,ou seja, verdadeiro e falso.

2.6 - Tipos de Dados básicos do Java

Os tipos de dados mais utilizados são:

`byte`, `int` , `long`	- para números inteiros
`float` , `double`	- para números reais
`char` , `String`	- para valores caracter
`boolean`	- para valores lógicos (`true` ou `false`)

obs: todo valor do tipo String deverá ser escrito no programa entre áspas (" "), e valores do tipo char entre apóstrofe (‘ ‘). Isto é importante em um programa para diferenciar os valores do tipo String, char, e os demais tipos.

Tipos de dados mais básicos da Java:

Tipo de dado	Valores que podem assumir	Tamanho em bytes na memória
`byte`	de –128 até 127	1
`int`	de –2.147.483.648 a 2.147.483.647	4
`long`	de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	8
`float`	de -3.4×10^{38} a 3.4×10^{38}	4
`double`	de -1.7×10^{308} a 1.7×10^{308}	8
`char`	armazena apenas um caracter	1
`String`	armazena um conjunto de caracteres	cada caracter = 1 byte
`boolean`	`true` ou `false`	1 bit

Tipos primitivos de dados: Com exceção do tipo String, os demais tipos de dados relacionados acima são considerados **tipos primitivos**. Os tipos primitivos são usados por variáveis que irão armazenar os dados mais simples (Números, caracteres da tabela ASCII e os valores lógico true ou false). O tipo String é na verdade uma Classe do Java, e portanto é bem mais complexa e utiliza métodos próprios para manipulação dos valores armazenados. Com isto, a forma de utilização e manipulação de valores String é diferente da forma usada com os tipos primitivos. Ao longo dos capítulos estas diferenças serão apresentadas.

2.7 – Declaração de Variáveis no Java

Toda variável, **para ser utilizada, deve primeiro ser declarada**. Ou seja, só é possível utilizar uma variável em uma linha de código que esteja depois da linha de declaração da variável.

No Java as variáveis **podem ser declaradas em qualquer parte do programa**, bastando para isto escrever em uma linha de comando o tipo da variável e em seguida o nome da variável. Podem ser declaradas dentro das classes ou métodos.

No entanto, por uma questão de organização e clareza de código**, a boa prática orienta que as variáveis devem ser declaradas sempre no inicio do método ou classe as quais serão utilizadas;**

Variáveis declaradas dentro de um método são consideradas **variáveis locais**, ou seja, só são válidas somente dentro daquele método. Variáveis declaradas em uma classe são consideradas **globais**, ou seja, podem ser utilizadas por quaisquer métodos existentes dentro daquela classe.

Exemplo:

```
public class ClasseExemplo1{
     public static void main (String[] args) {
          int idade;
          float salario, peso, altura;
          String nomeAluno;
          char sexo;
          boolean alunoFoiAprovado;
     }
}
```

2.8 – Atribuição de valores as Variáveis

Para se atribuir uma valor a uma variável utiliza-se o **comando de Atribuição** do Java (**=**).

Para atribuir um caracter a uma variável do tipo **`char`** utiliza-se o **apóstrofe** (‘), entretanto, quando se trata de um conjunto de caracteres atribuídos a uma variável do tipo **`String`**, se utiliza **aspas** (“).

Na linha de comando para atribuição de valores a uma variável, a variável sempre deverá ser escrita a esquerda do comando de atribuição (`=`) e o valor a ser atribuído escrito a direita.

O uso destes do comando de atribuição (`=`) indica que após a execução do comando, o valor escrito a direta do comando será inserido na variável escrita a esquerda do comando.

Obs: entre o nome da variável, o comando de atribuição e o valor, não é obrigatório o uso de espaço para separar cada um. Entretanto, por questões de clareza do código, é uma boa prática sempre utilizar 1 espaco antes e 1 espaço depois do comando de atribuição, como nos exemplos abaixo.

Exemplos:

```
idade = 25;                                 // tipo int
salario = 2786.50;                          // tipo float
nomeAluno = "Marcelo dos Santos";           // tipo String
sexo = 'F';                                 // tipo char
alunoFoiAprovado = true;                    // tipo boolean
nota1 = 3;                                  // tipo inteiro
nota2 = 4;                                  // tipo inteiro
notaFinal = nota1 + nota2;                  // tipo inteiro
notaProva = 25.5 + 35.2;                    // tipo float
notaLTP = nota1 + 10;                       // tipo inteiro
```

Obs: na oitava linha dos exemplos acima, a variável `notaFinal` recebeu a operação `nota1 + nota2`. Quando o comando de atribuição é utilizado em conjunto com uma operação (neste caso soma) o valor que será atribuído a variável será o resultado da operação. Desta forma, como `nota1` recebeu o valor `3` e `nota2` recebeu o valor `4`, `notaFinal` receberá o resultado da soma de `3 + 4`, ou seja, o valor `7` será atribuído a variável `notaFinal`. Nas duas linhas seguintes, `notaProva` vai receber

o resultado da soma de `25.5` com `35.2`, que será `60.7`. E `notaLTP` vai receber o resultado da soma do valor contido na variável `nota1` (`3`) com `10`, que será `13`.

2.9 – Linha de Comando no Java

Uma linha de comando no Java pode ocupar uma ou mais linhas do código fonte, e sempre se encerra com ponto-e-vírgula (;). Há também a possibilidade de se escrever mais de uma linha de comando em uma mesma linha do programa fonte, mas **esta prática deve ser evitada** pois dificulta a interpretação da lógica e a identificação da existência de mais de um comando na respectiva linha de código.

Exemplo:

```
salario = 2786.50;     //uma linha de comando em uma linha de programa fonte
nomeDoAluno = "Marcelo dos Santos";
sexo = 'F'; aprovado = true; //2 comandos em uma linha, evitar isto !!
salario = salario + tempoCasa – descontoValeTransp + 5 / 100 * 2 +
      2786.50;   // 1 comando em duas linhas do programa fonte
texto = "O Java é uma linguagem de programação orientada a objeto que visa"
     + " ampliar a produtividade"; // 1 comando em duas linhas do programa fonte
```

Obs: segundo os padrões de codificação Java definidos pela Sun, um linha de comando não deve ultrapassar 80 caracteres, facilitando a visualização do fonte mesmo em monitores com baixa resolução. Neste caso, o comando deve continuar na linha seguinte, mas sempre identado em relação à primeira linha do comando, como no ultimo exemplo acima.

3 – Operadores

Os operadores são utilizados para manipulação de dados somente em variáveis de tipos primitivos.

3.1 - Operadores Aritméticos

Os operadores aritméticos são utilizados para efetuar operações aritméticas com número inteiros e reais. São eles:

Operador	O que representa	Exemplos de uso
+	**Soma**	**2 + 3 b + c x + 20 + 2**
-	**Subtração**	**10 - 2 b – a y - 3 - x**
*****	**Multiplicação**	**3 * 4 x * y a * 2 * b**
/	**Divisão**	**x / y 10 / 2 10 / x / b**
%	**Resto da divisão**	**10 % 3 // resultado será 1**
--	**decremento de 1**	**a-- // mesmo que a = a - 1**
++	**Incremento de 1**	**a++ // mesmo que a = a + 1**
+=	Soma com atribuição	a += b // mesmo que a = a + b
-=	Subtração com atribuição	x -= y // mesmo que x = x - y
*=	Multiplicação com atribuição	b *= c // mesmo que b = b * c
/=	Divisão com atribuição	x /= y // mesmo que x = x / y
%=	Resto da divisão com atribuição	a %= b // mesmo que a = a % b

= ++	Incremento de 1 com atribuição (equivale a dois comandos)	b = ++a // mesmo que a = a + 1
		// em seguida b = a

= ++	Atribuição com incremento de 1 (equivale a dois comandos)	b = a++ // mesmo que b = a
		// em seguida a = a + 1

= --	decremento de 1 com atribuição	b = --a // mesmo que a = a - 1

	(equivale a dois comandos)	`// em seguida b = a`
`= --`	Atribuição com decremento de 1 (equivale a dois comandos)	`b = a--  // mesmo que b = a` `// em seguida a = a - 1`

3.1.1 - Resolução de operação aritmética composta

Em uma operação matemática composta de vários operadores aritméticos, a ordem de resolução é:

1°. → o que está entre parênteses
2°. → operações de multiplicação, divisão e resto da divisão
3°. → operações de soma e subtração

Exemplo:

```
Result = (2 + 4) - 6 / 2 + 5 * (2 + 3) + 9 % 2;   // 1ª etapa
          ------                ------
Result =    6    - 6 / 2 + 5 *    5    + 9 % 2;   // 2ª etapa
                   -----   --------      -----
Result =    6    -   3   +    25       +   1;     // 3ª etapa
           ---------------------------------
Result =                  29;
```

3.1.2 - Divisão entre valores Inteiros no Java

Em Java, toda divisão entre inteiros gera um resultado também inteiro, ou seja, as casas decimais são eliminadas (truncadas). Desta forma, uma divisão só retornará valor real (`double` ou `float`) contendo as respectivas casas decimais,se pelo menos um dos valores envolvidos na operação for do tipo real. Então, se o objetivo for obter um o resultado real em uma divisão entre dois valores inteiros, deverá ser atribuído no momento do cálculo um valor real para um dos inteiros envolvidos. Está técnica recebe o nome de `typecasting`.

Exemplo 1:

```
int numPessoas = 21;
int somaIdades = 200;
float mediaIdades = somaIdades /numPessoas;
```

No Exemplo acima, para calcular o valor da média de idades, será feita a divisão `somaIdades / numPessoas` e o resultado que será atribuído a variável `mediaIdades` será o valor inteiro `9` ao invés de `9.5238`, que seria o resultado real da divisão. Ou seja, como os operandos`somaIdades`e `numPessoas` são valores inteiros, o resultado da operação também uma valor inteiro, independentemente da variável que está recebendo este resultado (`mediaIdades`) ser do tipo `float`.

Exemplo 2:

```
int numPessoas = 21;
int somaIdades = 200;
float mediaIdades = (float) somaIdades /numPessoas;
```

Neste segundo exemplo foi inserida na operação um tipo `float` para a variável `somaIdades`.Isto indicará ao compilador que apesar da variável `somaIdades` ser do tipo `int`, no momento da operação ela deverá ser considerada como um `float`.E neste caso, como um dos valores é do tipo real (`double`

ou `float)` o resultado da divisão também será um valor real. Ou seja, o valor que será atribuido a variável `mediaIdades` será `9.5238` .

3.2 - Operadores Relacionais

Os operadores relacionais podem ser utilizados para comparar valores primitivos da mesma classe (as classes são: numéricos, alfanuméticos e lógicos). Mas são mais usados na comparação entre valores numéricos. No entanto, não é possível utilizar este operadores para comparar valores de classes diferentes. Desta forma, não é possível comparar valores numéricos com alfanuméricos, ou numéricos com lógicos, ou alfanuméricos com lógicos. Os operadores relacionais são:

Operador	O que representa
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
!=	Diferente de
==	Igual a

O resultado de uma operação relacional é sempre um valor lógico, ou seja, `true` ou `false`.

Exemplos: considere as variáveis e expressões abaixo:

```
int  notaA = 1;
byte notaB = 3;
float notaC = 5.3;
char classe1 = 'A';
char classe2 = 'B';
boolean resultado;

resultado = notaC == notaA;     // resultado será false
resultado = notaA > notaB;       // resultado será false
resultado = notaB <= notaC;     // resultado será true
resultado = notaA != notaB;     // resultado será true
resultado = classe2 > classe1;  // resultado será true
```

Obs: se duas variáveis do tipo `char` forem comparadas por meio de um operador relacional, será considerada a grandeza sequencial do caracter da tabela ASCII para se obter o resultado da comparação. No exemplo acima, a variável `classe2` foi inicializada com o valor `'B'` e variável `classe1` com o valor `'A'`. Na tabela ASCII o caracter B aparece após o A, ou seja, o caracter B é considerado então maior que a o A em uma comparação relacional.

3.3 - Operadores Lógicos

Os operadores lógicos são utilizados para se criar expressões Relacionais compostas. São eles:

Operador	O que representa
&&	e (junção)
||	ou (escolha)
!	não (negação ou inversão)

Tabela verdade para operações lógicas:

Abaixo, segue três tabelas, chamadas tabela-verdade, contendo o resultado douso dos operadores lógicos sobre dois operandos.

Operando 1	Operador	Operando 2	Resultado
true	e	true	true
true	e	false	false
false	e	false	false
true	ou	true	true
true	ou	false	true
false	ou	false	false
-------	não	true	false
-------	não	false	true

Exemplos: considere as variáveis e expressões abaixo

```
int  notaA = 1;
byte notaB = 3;
float notaC = 5.3;
Boolean resultado;

resultado = ( notaA > notaB  &&  notaB <= notaC );
              -------------      --------------
//               false       e      true         =   false

resultado = ( notaA != notaB  ||  notaC < notaB );
              --------------     --------------
//                true       ou      false       =   true

resultado = ( !  notaB <= notaC );
              ---------------
//           não     true          =  false
```

Obs: no Java, todas os testes condicionais devem estar dentro de parênteses, os demais parenteses separando as partes de uma expressão composta são opcionais. No entanto, esta separação é recomendada segundo os padrões de codificação Java da Sun Microsystem quando é necessário facilitar o entendimento das prioridades da execução.

3.4 - Prioridade na resolução de operações compostas

Se vários operadores aparecerem em uma expressão, a ordem de execução dasoperações será dadasegundo os seguintes critérios seqüenciais:

1º.	Parênteses →( )
2º.	Métodos (funções) → Ex: Math.round() , Math.pow(), ...
3º.	Multiplicação, Divisão e Resto da Divisão→ * , / , %
4º.	Soma e Subtração → + , -
5º.	Operadores Relacionais, sem ordem entre eles → **>** , **<** , **>=** , **<=** , **!=** , **==**
6º.	Operadores Lógicos na seguinte ordem: não, e, ou → ! , && , ||

3.5 - Exercício – Operações compostas.

Resolver as operações compostas abaixo apontando o resultado final:

Questão 1.1

```
a) 1 < 3 && 20 / 2 == 10
b) 1 < 3 || 20 / 2 == 10
c) 7 == 6 || 21 / 3 != 18 / 3
d) ! 10 >= 9 || 3 * 3 / 2 > 20 - 5
e) ! (5 != 10 / 2) || 2 <= 3
```

Questão 1.2 – considerando as variáveis `A = 2, B = 5 , C = 2.5, X = false, Y = true`, resolva as expressões abaixo:

```
a) X && B / A >= C || ! A <= C
b) ! X || Y && (A + B) * C <= A
c) B / A == C || X && C > B
d) Y || A * B > C && C * 10 – A + B > B + C
```

3.6 - Operador de Concatenação de String (+)

O operador de concatenação (**+**) permite a junção de valores do tipo `String` com outra `String`, ou valores do tipo `String` com `char`, ou valores do tipo `String` com `int (ou outro tipo numérico)`. O resultado da junção será sempre um valor do tipo `String`.

Exemplo:

```
public class ExemploConcatenacao {
     public static void main(String[] args) {
          String nome = "Josias";
          String sobreNome = "Santos";
          String nomeCompleto;
          Int idade = 20;
          String data;
          nomeCompleto = nome + ' ' + sobreNome;
          System.out.println( nomeCompleto );    // Josias Santos
          nomeCompleto = "Jose" + ' ' + "Maria";
          System.out.println( nomeCompleto );    // Jose Maria
          nomeCompleto = nome + idade;
          System.out.println( nomeCompleto );    // Josias20
          data = idade + "/02/2024";
          System.out.println( data );    //  20/02/2024
     }
}
```

4 – Comandos de Entrada e Saída de dados na console

4.1 - Saída de dados

Um dos principais comandos destinadas a exibir os dados na console de uma IDE (Integrated Development Environment):

`System.out.println` e `System.out.print`

Sintaxe:

```
System.out.print( variável1 + variável2 + expressão1 + expressão2 + …);
System.out.println( variável1 + variável2 + expressão1 + expressão2 + …);
```

A diferença entre `System.out.print` e `System.out.println` é que no primeiro, o cursor se posiciona na tela logo a frente ao último valor exibido (exemplo 1). Já no caso do `System.out.println`, o cursor se posiciona na primeira coluna da próxima linha (exemplo 2).

Exemplo1:

```
System.out.print("TESTE DE TELA");
```

```
TESTE DE TELA_
```

Exemplo2:

```
System.out.println("TESTE DE TELA");
```

```
TESTE DE TELA
_
```

Quando se utiliza estes comandos de saída de dados com variáreis do tipo real (`float ou double`), para evitar a exibição de número excessivo de casas decimais, pode-se fazer a formatação dos dados de saída em relação ao número destas casas decimais. Para formatar decimais no Java pode-se utilizar a classe `DecimalFormat` que está contida no pacote de classes `text`, que deverá ser importado no início do programa `(import java.text.*;)`.

4.2 – Entrada de Dados

Existe mais de uma forma de realizar entrada de dados no Java. Uma delas é utilizando a classe `Scanner`. A classe `Scanner` está contida no pacote `útil`, então é necessário importá-lo no início do programa `(import java.util.*;)` .Neste caso, todas as entradas de dados são recebidas pelo Java como uma sequencia de caracteres, que deverão ser convertidos utilizando-se funções de conversão de tipos do Java:

Método	Descrição
`next()`	Aguarda a digitação de um valor do tipo `String` com uma palavra (sem espaços)
`nextLine()`	Aguarda a digitação de valor do tipo `String`, com uma ou mais palavras
`next().charAt(0)`	Aguarda a digitação de valor do tipo `char` com apenas um caracter
`nextInt()`	Aguarda a digitação de um valor do tipo `int`
`nextLong()`	Aguarda a digitação de um valor do tipo `long`
`nextByte()`	Aguarda a digitação de um valor tipo `byte`
`nextFloat()`	Aguarda a digitação de um valor tipo `float`
`nextDouble()`	Aguarda a digitação de um valor tipo `double`

Exemplo 1 - Código Java (classe) com declaração de variáveis, entrada e saída de dados:

```
import java.util.*;
public class ExemploEntrada {
  public static void main(String[] args) {

    // 1 - declaração das variáveis
    String nome;
    char sexo;
    float salario;
    byte idade;

    // 2 - entrada dos dados
    Scanner leia;          // declara a variável leia  para ser utilizada na classe Scanner
    leia = new Scanner(System.in);   /* inicializa a variável leia  para receber os
                                        valores digitados na de entrada de dados */
    System.out.print("Digite o Nome: ");
    nome = leia.nextLine();      // recebe o valor digitado e armazena na variável NOME

    System.out.print("Digite o Salario: ");
    salario = leia.nextFloat();   //  recebe o valor digitado e armazena na var. SALARIO

    System.out.print("Digite a Idade: ");
    idade = leia.nextByte();      //  recebe o valor digitado e armazena na variável IDADE

    System.out.print("Digite o Sexo: ");
    sexo = leia.next().charAt(0); //recebe o valor digitado e armazena na var. sexo

    // 3 - Saída de dados
    System.out.println("O nome digitado foi: " + nome);
    System.out.println("O salario digitado foi: " + salario);
    System.out.println("A idade digitada foi: " + idade);
    System.out.println("O sexo digitado foi: " + sexo);
  }
}
```

Exemplo 2 (fazer no Eclipse):

Faça um programa em Java que receba via teclado o Salário e o Valor do Aumento Salarial de um empregado. Em seguida o programa deverá calcular e imprimir o novo salário do empregado, que será a soma do atual salário com o Valor do Aumento Salarial.

```
import java.util.*;
public class Exemplo {
     public static void main(String[] args) {

          // 1 - declaração das variáveis
          float salario;
          float vlrAumento;
          float novoSal;

          // 2 - entrada dos dados
          Scanner leia;
          leia = new Scanner(System.in);
          System.out.print("Digite o Salário: ");
          salario = leia.nextFloat();
          System.out.print("Digite o Valor do Aumento: ");
          vlrAumento = leia.nextFloat();

          // 3 - cálculo
          novoSal = salario + vlrAumento;

          // 4 - Saída de dados
            System.out.print("Novo salário:" + novoSal );

     }
}
```

EXERCÍCIOS (parte 1):

Exercício 1.1 - Fazer um programa em Java que receba via teclado o Nome de um Aluno, e as três notas que ele tirou nas três avaliações de uma disciplina (Nota 1, a Nota 2 e a Nota 3). Em seguida programa deverá calcular e exibir a nota final do aluno na disciplina e média de notas por avaliação.

Exercicio 1.2 - Fazer um programa em Java que receba via teclado o Nome e o Salário de um empregado. Em seguida o programa deverá calcular e exibir o Novo Salário do empregado considerando que ele recebeu um aumento de 15%.

Exercicio 1.3 - Fazer um programa em Java que receba via teclado o Valor em dinheiro a ser colocado em uma aplicação financeira e o valor da Taxa de Juros a ser aplicada. Em seguida o programa deverá calcular e exibir o Valor recebido como rendimento da aplicação e o Valor total após o rendimento.

Exercicio 1.4 - Faça um programa em Java que receba via teclado a distância em km e o tempo em horas de um veículo. Calcule e imprima a velocidade deste veículo em km/h.

Exercício 1.5 – Considerando que:
1 pé = 12 polegadas;
1 polegada = 2,54 centímetros;
1 jarda = 3 pés;
1 milha = 1,760 jarda
Faça um programa em Java que receba via teclado a digitação de uma medida em pés, em seguida, faça as conversões e exiba o valor em polegadas, centímetros, jardas e milhas.

Exercicio 1.6 – Um empregado depositou todo o salário recebido no mês em uma conta bancária. O empregado emitiu dois cheques. O banco cobra uma taxa de 0,02 % do valor do cheque para cada cheque emitido. Considerando que antes do depósito o saldo na conta era zero, faça um programa em Java que receba via teclado a digitação do valor Depositado e os valores dos dois Cheques emitidos. Em seguida o programa deverá calcular e exibir o saldo final na conta bancária após a compensação dos cheques e das taxas por cheque.

5 – Estruturas de alternativa (condicional)

Situações condicionais fazem parte do cotidiano de todas as pessoas. Frequentemente nos vemos em situações onde devemos tomar uma decisão ou executar uma ação se uma condição ocorrer, por exemplo:

```
se vai chover então vou levar o guarda-chuva
```

neste exemplo a condição é: `vai chover` , e caso esta condição seja verdadeira, ou seja, a chuva aconteça, a ação a ser tomada é `levar o guarda-chuva`.

Em um algoritmo, uma estrutura condicional deve ser utilizada quando a execução de uma linha de código depende de uma condição.

As estruturas condicionais são divididas em 2 tipos:

→ Condição Simples
→ Condição Composta

5.1 - Identação

Identação é o recuo de um texto em relação a sua margem (tabulação). Na construção de programas de computadores, utilizamos a identação para ressaltar a estrutura do programa, organizar o código e indicar grupos e subgrupos de comandos, o que facilita a interpretação do código escrito pelo programador. A identação irá indicar com clareza que um conjunto de comandos pertence a uma certa estrutura de código. Isto engrandece a qualidade do código escrito e, apesar da identação não ser obritatória, um código escrito sem identação ficará confuso e de difícil interpretação por um programador.

Nesta disciplina utilizaremos as regras de identação presentes nas boas práticas de programação em Java definidas pela Sun Microsystem (atualmente Oracle) para a construção de algoritmos em Java. Estas boas práticas de programação foram criadas com o objetivo de definir um padrão de codificação para a linguagem de forma que qualquer programador possa interpretar com facilidade um código escrito em Java.

Um código de boa qualidade escrito por um programador experiente perderá qualidade se não tiver seguido as boas práticas de programação.

Nas sintaxes das estruturas condicionais a seguir serão indicadas as identações que devem ser utilizadas para ilustrar e exemplificar este conceito.

5.1 – Estrutura de Alternativa Simples (`if`)

Sintaxe:

Um comando por alternativa	Mais de um comando por alternativa

`if ( condição ) {`
`  comando1;`
`}`
	`if ( condição ) {`
	`  comando1;`
	`  comando2;`
	`  comando3;`
	`}`

Obs: quando existir apenas uma linha de comando dentro da estrutura de alternativa, não é necessário que este comandos esteja entre chaves - `{ }`, no entanto, as boas práticas de codificação Java da Sun Microsystem indicam que as chaves devem ser sempre utilizadas.

Identação: Na sintaxe acima a identação está presente no recuo utilizado em cada linha de comando existente entre o `if` e a chave final `}`. Como forma de facilitar a escrita do programa, em cada espaço de identação é utilizado um nível de tabulação no editor de programa (tecla TAB). Uma tabulação é suficientes para indicar visualmente o recuo e indicar que os comandos escritos com esta tabulação estão embutidos dentro da estrutura condicional, ou seja, só serão executados se a `condição` for verdadeira.

Repare que o fechamento da chave `}` deverá ser colocado na mesma coluna da letra `i` do comando `if`, ou seja, os dois ficarão alinhados verticalmente no código do programa. O mesmo acontece com os comandos dispostos entre o `if` e a chave final `}`, todos estão alinhados por meio da tabulação e com a digitação iniciando na mesma coluna do editor de texto.

Exemplo 1:

```
int varA = 10;
int varB = 5;
if (varA > varB) {
     System.out.print("A é maior que B !");
}
```

No exemplo acima o valor que foi atribuído a variável `varA` é maior que o valor atribuído a variável `varB`. Então, a condição `varA > varB` é verdadeira e o texto `A é maior que B !` será exibido na tela.

Exemplo 2:

```
byte idade;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a idade: ");
idade = leia.nextByte();
if (idade < 2) {
     System.out.print("É um bebê ! ");
}
```

No exemplo acima o valor que será atribuído a variável `idade` será digitado pelo usurário. Caso o valor digitado seja menor que 2, a condição `idade < 2` será verdadeira e o texto `É um bebê !` será exibido na tela. Mas se o valor digitado pelo usuário for maior ou igual a 2, a condição `idade < 2` será falsa e nenhum comando será executado.

Exemplo 3:

```
byte idade;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a idade: ");
idade = leia.nextByte();
if (idade < 2) {
     System.out.print("É um bebê ! ");
     idade = idade + 5;
}
```

No exemplo acima o valor que será atribuído a variável `idade` será digitado pelo usurário. Caso o valor digitado seja menor que 2, a condição `idade < 2` será verdadeira, o texto `É um bebê !` ser exibido na tela, mas em seguida, no próximo comando, o valor da variável `idade` será incrementado em `5`. Ou seja, se o usuário, por exemplo, tiver digitado o valor 1 para a variável `idade`, após exibir a

mensagem `É um bebê !` , a variável `idade` passará a conter o valor `6` (`1 + 5`). Mas se o valor digitado pelo usuário for maior ou igual a 2, nenhum comando será executado.

5.2 – Estrutura de Alternativa Composta (`if...else`)

Sintaxe:

Um comando por alternativa	Mais de um comando por alternativa

`if ( condição ) {`
`  comando1;`
`} else {`
`  comando2;`
`}`
	`if ( condição ) {`
	`  comando1;`
	`  comando2;`
	`} else{`
	`  comando3;`
	`  comando4;`
	`}`

Na condição composta, se a `condição` for verdadeira, serão executados somente os comandos que estão dentro do das chaves do comando `if`,mas se a `condição` for falsa, serão executados somente os comandos que estão dentro das chaves do comando `else`.

Identação: Na sintaxe acima a identação está presente no recuo utilizado em cada linha de comando existente entre o `if` e a chave final `}` e entre o `else` e a segunda chave final `}`.

Exemplo 1:

```
byte idade;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a idade: ");
idade = leia.nextByte();
if (idade < 2) {
     System.out.print("É um bebê ! ");
} else {
     System.out.print(" Não é um bebê ! ");
}
```

No exemplo acima o valor que será atribuído a variável `idade` será digitado pelo usurário. Caso o valor digitado seja menor que 2, a condição `idade < 2` será verdadeira e o texto `É um bebê !` será exibido na tela. Mas se o valor digitado pelo usuário for maior ou igual a 2, a condição `idade < 2` será falsa e o texto `Não é um bebê !` será exibido na tela.

Exemplo 2:

```
float nota;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a nota do aluno: ");
nota = leia.nextFloat();
if (nota >= 60) {
     System.out.print("O aluno está aprovado! ");
} else {
     System.out.print("O aluno está reprovado!");
}
```

Neste exemplo, se a nota digitada for maior ou igual a 60, será executado somente o comando escrito entre o `if` e o primeiro fecha-chave `}`:

```
System.out.print("O aluno está aprovado! ");
```

Mas se a nota digitada for menor que 60, será executado somete o comando escrito entre o `else` e o segundo fecha-chave `}`:

```
System.out.print("O aluno está reprovado!");
```

7.4 - Estrutura Condicional Composta Aninhada

Sintaxe:

Mais de uma alternativa (else if)	

`if ( condição1 ) {`
`  comando1;`
`  comando2;`
`} else if ( condição2 ) {`
`  comando3;`
`}`
	`if ( condição1 ) {`
	`  comando1;`
	`  comando2;`
	`} else if ( condição2 ) {`
	`  comando3;`
	`} else if ( condição3 ) {`
	`  Comando4;`
	`  Comando5;`
	`} else {`
	`  Comando6;`
	`}`

Exemplo1:

Criar uma lógica condicional para o problema a seguir:

NOTA	SITUAÇÃO
de 60 para cima	aluno aprovado
de 40 até 59	aluno em recuperação
abaixo de 40	aluno reprovado

```
byte nota;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a nota do aluno: ");
nota = leia.nextByte();
if (nota < 40) {
     System.out.print("Aluno reprovado");
} else if (nota <= 59) {
     System.out.print("Aluno em recuperação");
} else {
     System.out.print("Aluno aprovado");
}
```

Neste exemplo acima, se a nota digitada for menor que 40, será executado somente o comando escrito entre o primeiro `if` e a primeira fecha-chave `}` e em seguida a estrutura de alternativa é finalizada.

```
System.out.print("Aluno reprovado");
```

Mas se a nota digitada for maior ou igual a 40, o primeiro `if` será falso e então o segundo `if` será testado. Repare que no segundo `if` não é necessário testar se a nota é maior ou igual a 40, porque se o segundo `if` está sendo testado, é porque o primeiro `if` foi falso, ou seja, o segundo `if` só será executado se a nota for maior ou igual a 40. Desta forma, só há necessidade de testar se a nota é menor

ou igual a 59. Sendo assim, neste segundo teste, se a nota digitada for maior ou igual a 40 e menor ou igual a 59, o comando abaixo será executado:

```
System.out.print("Aluno em recuperação");
```

Mas se a nota digitada for maior que 59, os dois primeiros testes serão falsos e apenas o comando escrito no ultimo `else` será executado:

```
System.out.print("Aluno aprovado");
```

Ou seja, em uma estrutura de alternativas aninhadas com vários `if` e `else`, independente do número de alternativas que existirem, somente serão executados os comandos contidos dentro das chaves cuja alternativa for verdadeira. Os testes sempre obedecem a ordem de cima para baixo em que foram escritas as alternativas. Se existirem mais de uma alternativa verdadeira, somente os comandos da primeira que for verdadeira serão executados, considerando a ordem em que foram escritas.

Exemplo2:

Criar uma lógica condicional para o problema a seguir:

idade	sexo	resultado
0 ou 1 ano	M	Bebê menino
0 ou 1 ano	F	Bebê menina

```
byte idade;
char sexo;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a idade: ");
idade = leia.nextByte();
System.out.print("Digite o sexo (M/F): ");
sexo = leia.next().charAt(0);
if (idade < 2 && sexo == 'M'){
     System.out.print("É um bebê menino ! ");
} else if (idade < 2 && sexo == 'F') {
     System.out.print(" É um bebê menina ! ");
}
```

No exemplo acima os valorer que serão atribuídos as variáveis `idade` e `sexo` serão digitados pelo usurário. Caso o valor digitado para `idade` seja **menor que 2 e** o valor digitado para `sexo` for a letra **M**, a condição `idade < 2 && sexo == 'M'` será verdadeira e o texto `É um bebê menino !` será exibido na tela. Mas se o valor digitado pelo usuário para `idade` for menor que 2 e o valor digitado para `sexo` for a letra **F** , a segunda condição `idade < 2 && sexo == 'M'` será verdadeira e o texto `É um bebê menina !` será exibido na tela. Em quaisque outros valores atribuídos as variáveis `idade` e `sexo` nenhuma das duas condições testadas serão verdadeira e, portanto, nenhum comando será executado.

Exemplo3:

Criar uma lógica condicional para o problema a seguir:

idade	resultado
0 ou 1 ano	Bebê
2 a 9 anos	Criança
10 a 15 anos	Adolescente
16 anos em diate	Adulto

```
byte idade;
Scanner leia = new Scanner(System.in);
System.out.print("Digite a idade: ");
idade = leia.nextByte();
if (idade < 2) {
     System.out.print(" É um bebê ! ");
} else if (idade < 10) {
     System.out.print(" É uma criança ! ");
} else if (idade < 16) {
     System.out.print(" É um adolescente ! ");
} else {
     System.out.print(" É um Adulto ! ");
}
```

5.3 - Comando de Alternativa *Switch*.

Diferentemente da estrutura `if-then` e `if-then-else`, a esturura `switch` pode ter diferentes caminhos de execução. Switch trabalha com os tipos primitivos `byte`, `short`, `char` e `int` e com a classe `String`.

No exemplo a seguir é declarada uma variável do tipo `int` de nome mes, cuja o valor representa o numeral do mes. O código exibe o nome do mes de acordo com o valor por meio de uma estrutura switch.

```
public class Exemplo Switch {
    public static void main(String[] args) {
        int mes = 8;
        String mesString;
        switch (mes) {
            case 1:  mesString = "Janeiro";
                     break;
            case 2:  mesString = "Fevereiro";
                     break;
            case 3:  mesString = "Março";
                     break;
            case 4:  mesString = "Abril";
                     break;
            case 5:  mesString = "Maio";
                     break;
            case 6:  mesString = "Junho";
                     break;
            case 7:  mesString = "Julho";
                     break;
            case 8:  mesString = "Agosto";
                     break;
            case 9:  mesString = "Setembro";
                     break;
            case 10: mesString = "Outubro";
                     break;
            case 11: mesString = "Novembro";
                     break;
            case 12: mesString = "Dezembro";
                     break;
            default: mesString = "Mes inválido";
```

```
                    break;
            }
            System.out.println(mesString);
        }
}
```

Neste caso, o texto Agosto será exibido.

O corpo de uma estrutura Switch é chamada de bloco. A estrutura de um bloco switch pode ser criada com um ou mais opções "case", e uma opção "default". A estrutura switch verifica o valor da variável, caso seja equivalente a uma ou mais opções "case", todas as equivalentes serão executadas. A opção "default" é executada se não houver equivalência a nenhuma opção "case".

O exemplo anterior também poderia ser feito utilizando a estrutura `if-then-else`:

```
int mes = 8;
if (mes == 1) {
   System.out.println("Janeiro");
} else if (mes == 2) {
   System.out.println("Fevereiro");
}
...  // e assim por diante
```

Decidir quando utilizar `if-then-else` ou `switch`, vai depender da clareza desejada no código e do tipo de teste condicional do comando. O `if-then-else` permite testar condições envolvendo expressões lógicas compostas e complexas, enquanto o switch permite apenas testar o valor de uma única variável.

Outro opção interessante do `switch` é o uso do comando `break`. Cada `break` finaliza o estrutura `switch` no qual se encontra. Se o `break` não for utilzado, todas as alternativas `case` serão testadas e aquelas que forem verdadeiras serão executadas. Com a utilização do break, após a execução da primera alternativa `case` verdadeira, o `switch` é encerrado sem testar as demais alternativas.

No exemplo abaixo, por não ter o comando `break` nas alternativas `case` de 1 a 11, após a execução da alternativa `case 8:` , (que é a primeira verdadeira) o valor do mês terá sido incrementado de 1 e a alternativa `case 9:` será também verdadeira e executada, e assim por diante até o `break` encontrado na alternativa `case 12:`

```
public class ExemploSwitch2{
    public static void main(String[] args) {
        int mes = 8;
        switch (mes) {
            case 1:  System.out.println("Janeiro");
                     mes ++;
            case 2:  System.out.println("Fevereiro");
                     mes ++;
            case 3:  System.out.println("Março");
                     mes ++;
            case 4:  System.out.println("Abril");
                     mes ++;
            case 5:  System.out.println("Maio");
                     mes ++;
            case 6:  System.out.println("Junho");
                     mes ++;
            case 7:  System.out.println("Julho");
```

```
                    mes ++;
            case 8:  System.out.println("Agosto");
                    mes ++;
            case 9:  System.out.println("Setembro");
                    mes ++;
            case 10: System.out.println("Outubro");
                    mes ++;
            case 11: System.out.println("Novembro");
                    mes ++;
                    break;
            case 12: System.out.println("Dezembro");
            default: break;
        }
    }
}
```

Os valores exibidos na tela serão:

```
Agosto
Setembro
Outubro
Novembro
```

O exemplo a seguir utiliza a estrutura switch para calcular o número de dias de um determinado mes:

```
public class ExemploSwitch3 {
    public static void main(String[] args) {

        int mes = 2;
        int ano = 2016;
        int numDias = 0;

        switch (mes) {
            case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                numDias = 31;
                break;
            case 4: case 6: case 9: case 11:
                numDias = 30;
                break;
            case 2:
                if ( ((ano % 4 == 0) && !(ano % 100 == 0)) || (ano % 400 == 0))
                  numDias = 29;
                else
                    numDias = 28;
                break;
            default:
                System.out.println("Mes Inválido");
                break;
        }
        System.out.println("Número de dias do mes = " + numDias);
    }
}
```

O valor exibido na tela será: `Número de dias do mes = 29`

EXERCÍCIOS (parte 2):

Exercício 2.1 – Fazer um programa em Java que receba via teclado a idade de uma pessoa. Em seguida, se a pessoa possuir 18 ou mais anos, exibir a mensagem “Você é maior de idade”, caso contrário, exibir a mensagem “Você é menor de idade”.

Exercicio 2.2 – Faça um programa em Java que receba via teclado o Ano, Mês e Dia de nascimento de uma pessoa e o Ano, Mês e Dia atual. Em seguida, o programa deverá calcular e exibir a idade da pessoa.

Exercício 2.3 – Faça um programa em Java que receba via teclado o nome e o salário de um empregado da empresa ABC. O algoritmo deverá calcular e imprimir o novo salário para o funcionário de acordo com a tabela abaixo:

- Salário abaixo de R$1.000,00 – aumento de 15%
- Salário de R$1.000,00 para cima – aumento de 10%

Exercício 2.4 - Fazer um programa em Java que receba via teclado o Nome do Aluno, a Nota1, a Nota2 e a Nota3. O programa deverá calcular e imprimir a média e o desempenho do aluno baseado nas seguintes regras:

- média de 7 para cima => aluno aprovado
- média de 4 para baixo => aluno reprovado
- média entre 4.1 e 6.9 => aluno em recuperação

Exercício 2.5 – Fazer um programa em Java que receba via teclado o número de votos de 3 candidatos em duas seções eleitorais. Em seguida o programa deverá somar a quantidade de votos de cada candidato nas duas seções. Considerando que não há empate, o programa deverá verificar qual dos 3 candidatos foi o vencedor e exibir na tela.

Exercício 2.6 – Fazer um programa em Java que receba via teclado as medidas dos 3 lados de um triângulo. De acordo com os valores digitados o programa deverá imprimir o tipo do triângulo:

- não é triangulo, se a soma de dois lados for menor ou igual ao terceiro lado;
- equilátero, se os 3 lados forem iguais;
- isósceles, se 2 lados forem iguais e 1 diferente;
- escaleno, se os 3 lados forem diferentes;

Exercício 2.7 – Faça um programa em Java que receba via teclado o Peso (em kg) e a Altura (em metros) de um atleta. O algoritmo deverá calcular e imprimir:

- a situação da pessoa de acordo com a tabela de IMC (Índice de Massa Corporal) abaixo;
- os pesos ideal mínimo e ideal máximo para que a pessoa esteja dentro da faixa de peso ideal de acordo com a tabela de IMC (Índice de Massa Corporal) abaixo;

Tabela de IMC:

- IMC menor que 20 → pessoa está na situação **abaixo do peso**
- IMC entre 20 e 25 → pessoa está na situação de **peso ideal**
- IMC maior que 25 → pessoa está na situação **acima do peso**

Obs:
- para calcular o IMC utilizar a fórmula: ***IMC = Peso / Altura^2***
- para o peso ideal mínimo, utilizar a fórmula: ***Peso ideal mínimo = 20 * Altura^2***
- para o peso ideal máximo, utilizar a fórmula: ***Peso ideal máximo = 25 * Altura^2***

Segue um exemplo de dados para testar o programa:

Digite *Altura* = 1.75 e Peso = 80
→ *situação = acima do peso*
→ *Peso Mínimo* = 61.25 e *Peso Máximo* = 76.56

Exercício 2.8 – Faça um programa em Java para calcular e imprimir o Bônus Salarial e Desconto de Vale Transporte para um empregado da Empresa ABCDário LTDA. O programa deverá receber via teclado o Tempo de Casa e o Salário do Empregado e considerar:

Tempo de Casa	Salário	Bônus	Vale Transporte
Até 5 anos	Até R$300,00	R$50,00	5% do Salário
Até 5 anos	Acima de R$300,00	R$80,00	6% do Salário
De 6 a 10 anos	Até R$500,00	15% do Salário	R$70,00
De 6 a 10 anos	Acima de R$500,00 até R$2000,00	13% do Salário	R$90,00
De 6 a 10 anos	Acima de R$2000,00	12% do Salário	8% do Salário
Acima de 10 anos	----------	R$300,00	4% do Salário

6 – Estruturas de repetição - *loop*

As estruturas de repetição são utilizadas quando um conjunto de comandos precisa ser executado mais de uma vez. Sendo assim, ao invés de escrever várias vezes um mesmo comando, ele deverá ser escrito somente uma vez dentro de uma estrutura de repetição. A estrutura de repetição é que ficará encarregada em executar o comando várias vezes.

Exemplo 1:

Imagine que você vai solicitar a um usuário que digite o nome de 3 alunos. A forma incorreta para isto seria escrever 3 vezes o comando de entrada de dados:

```
Scanner leia;
leia = new Scanner(System.in);
System.out.print("Digite o nome do aluno 1: ");
nomeAluno = leia.nextLine();
System.out.print("Digite o nome do aluno 2: ");
nomeAluno = leia.nextLine();
System.out.print("Digite o nome do aluno 3: ");
nomeAluno = leia.nextLine();
```

A forma correta para as entradas de dados acima seria escrever uma única vez o comando dentro de uma estrutura de repetição a qual fará com que este comando seja executado 3 vezes:

```
Scanner leia = new Scanner(System.in);
int contadorAluno = 1;
while (contadorAluno <= 3) {
      System.out.print("Digite o nome do aluno : " + contadorAluno);
      nomeAluno = leia.nextLine();
      contadorAluno = contadorAluno + 1;
}
```

Neste exemplo acima os comandos `System.out.print()` e `leia.nextLine()` serão executados 3 vezes mas foram escritos somente 1 vez. Para controlar o número de repetições, foi criada uma variável `contadorAluno` que será incrementada a partir de 1 até 3 para cada execução dos comandos `System.out.print()` e `leia.nextLine()`.

As estruturas de repetição também são conhecidas nas linguagens de programação como *laço* ou *loop*.

6.1 – Estrutura de repetição *while* (enquanto...faça)

Caracteriza-se por verificar uma condição antes de entrar na repetição. Se a condição for verdadeira, os comandos contidos dentro da estrutura do loop serão executados repetidamente enquanto a condição permanecer verdadeira. Quando a condição se tornar falsa, os comandos que estão dentro do loop não serão mais executados e é encerrada a repetição, neste caso o programa segue executando os comandos após a repetição, ou seja, fora do loop.

Sintaxe:

Um comando dentro da repetição	Mais de um comando dentro da repetição

`while ( condição ) {`
`  comando1;`
`}`
	`while ( condição) {`
	`  comando1;`
	`  comando2;`
	`}`

Obs: quando existir apenas uma linha de comando dentro da estrutura de repetição `while`, não é necessário que este comandos esteja entre chaves - `{ }`, no entanto, é pelos padrões de codificação da Sun, as chaves devem ser utilizadas sempre.

Exemplo 1: Faça um algoritmo que imprima na tela os números de 1 a 15;

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      System.out.println("1");
      System.out.println("2");
      System.out.println("3");
      System.out.println("4");
      System.out.println("5");
      System.out.println("6");
      System.out.println("7");
      System.out.println("8");
      System.out.println("9");
      System.out.println("10");
      System.out.println("11");
      System.out.println("12");
      System.out.println("13");
      System.out.println("14");
      System.out.println("15");
   }
}
```


```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

No exemplo acima, os dois programas alcançarão o mesmo resultado, ou seja, exibirão na tela os números de 1 a 15. Mas a primeira forma é considerada errada porque o programador precisou escrever o mesmo comando várias vezes no código, o que dificulta a escrita e ocupa mais linhas do que necessário no código fonte.

Veja abaixo como seria a execução linha a linha (passo a passo) deste código:

Passo 1:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println(
      cont );
        cont = cont + 1;
      }
   }
```

```
}
```

Passo 2:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

Agora a condição do while é testada, e, neste momento ela é verdadeira

Passo 3:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

Em seguida, o comando de saída é executado e o valor 1 será exibido na tela

Passo 4:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

Na execução desta linha, o contador será incrementado em 1, e passará a valer 2

Passo 5:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;

      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;

      }
   }
}
```

Agora a execução chegou no fim do while } , então linha de execução volta para testar novamente a condição

Passo 6:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

condição do while é testada e continua verdadeira, já que o valor do **cont** é 2 nete momento.

Passo 7:

```
public class ExemploWhile1 {
   public static void main(String[] args) {
      byte cont = 1;
      while (cont <= 15) {
        System.out.println( cont );
        cont = cont + 1;
      }
   }
}
```

Em seguida, o comando de saída é executado e o valor 2 será exibido na tela

Passos seguintes continuam sucessivamente até que a condição do while se torne falsa!

Exemplo 2:

```
byte x = 1;
while (x <= 5) {
  System.out.println("O valor de x é: " + x );
  x = x + 1;
}
```

No exemplo acima, a mensagem `O valor de x é:` será exibida na tela e logo a frente dela o valores da variável `x` de `1` a `5`, como abaixo:

```
O valor de x é: 1
O valor de x é: 2
O valor de x é: 3
O valor de x é: 4
O valor de x é: 5
```

Exemplo 3: Dada a altura e a largura de um retângulo, calcular e imprimir a área. Em seguida, por 5 vezes incrementar 2 na altura e decrementar 1 na largura, e em cada uma das vezes calcular e imprimir novamente a área do retângulo.

```
int area;
int altura = 2;
int largura = 6;
int x = 1;
while (x <= 5) {
  area = altura * largura;
  System.out.println("A Área é: " + area);
  altura = altura + 2;
  largura --;
  x ++;
}
```

Segue abaixo a tabela de valores de cada variável passo a passo na execução da lógica de repetição acima:

Variáveis	Passo 0	Passo 1	Passo 2	Passo 3	Passo 4	Passo 5
area	Null	12	20	24	24	20
altura	2	4	6	8	10	12
largura	6	5	4	3	2	1
x	1	2	3	4	5	6

Na tabela acima, o passo 0 indica o valor das variáveis antes de testar a condição (`x <= 5`) da lógica de repetição. Os passos 1 em diante são após o teste da condição. No passo 5, o valor de X se tornou 6, então a condição do while será falsa e a lógica de repetição será finalizada.

Exemplo 4 (fazer no Eclipse):

Faça um programa em Java para calcular e imprimir a soma dos números pares entre si até um final que deverá ser informado pelo usuário.

Neste caso, vamos somar os números pares entre si até um valor informado. Supondo que o valor informado seja 20, a soma seria entre os números:

2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20

Planejamento do programa:

1 - Declaração de variáveis:
Vamos precisar de uma variável para guardar o número par (numeroPar), uma para receber o valor final informado pelo usuário (valorFinal) e uma para guardar a soma (somaDosPares)

2 – Entrada de dados:
Receber a digitação do valor final para a soma

3 – Cálculos:
Fazer a soma dos pares

4 – Saída de Dados:
Imprimir a soma calculada

```java
import java.util.*;

public class ExemploWhile4 {

   public static void main(String[] args) {

     // 1 - Declaração e inicialização de variáveis
     int numeroPar = 2;
     int valorFinal;
     int somaDosPares = 0;
     Scanner leia = new Scanner(System.in);

     // 2 - Entrada de dados
     System.out.print("Digite o valor final para a serie: ");
     valorFinal = leia.nextInt();

     // 3 - Cálculos
     while (numeroPar <= ValorFinal) {
        somaDosPares = somaDosPares + numeroPar;
        numeroPar = numeroPar + 2;
     }

     // 4 - Saída de dados
     System.out.print("A soma dos números pares entre 2 e " +
           valorFinal + " é: " + somaDosPares);
   }
}
```

6.2 - O comando *break* e o uso de *flag* para interromper a estrutura de repetição.

O comando **break** interrompe uma estrutura de repetição, fazendo com que a execução do programa prossiga a partir da primeira linha após o final do comando de repetição.

O *flag* é uma informação chave a ser digitada na entrada de dados pelo usuário para indicar que não há mais dados a serem digitados. Esta chave indica então que a repetição do comando de entrada de dados deve ser encerrada.

O *flag* é util em situações onde o número de informações a ser digitada pelo usuário é variável, ou não previsível, e por isto não há como criar uma condição pré-definida para encerrar a entrada de dados. Como por exemplo o uso de um contador para controlar que a repetição ocorra um determinado número de vezes.

Desta forma, o flag indicará o momento de finalizar uma repetição por meio do comando **break**.

Exemplo:

```
int idade;
String nome;
int quantEmpregados = 0;
while (quantEmpregados <= 50) {
  System.out.print("Digite a idade do empregado (zero p/ fim): ");
  idade = leia.nextInt();
  if (idade == 0) {
    break;
  }
  System.out.print("Digite o nome do empregado: ");
  nome = leia.nextLine();
  quantEmpregados ++;
}
System.out.print("A quantidade de empregados é: " + quantEmpregados);
```

Neste exemplo acima, apesar do comando `while` ter uma condição que se tornará falsa quando a quantidade de empregados atingir 51, se o valor digitado na variável `idade` for `0` (zero), o `while` será finalizado pelo comando `break`, e a próxima linha de comando a ser executada será a que existe logo após o fecha-chaves `}` final do `while`, que é a exibição na tela da quantidade de empregados contida na variável `quantEmpregados;`

EXERCÍCIOS (parte 3):

Exercício 3.1 - Fazer um programa em Java para receber via teclado a nota de 10 alunos da disciplina LTP. Considerando que para ser aprovado na disciplina os alunos devem ter notas iguais ou maiores que 60, após a digitação de cada nota, exibir na tela se o aluno está Aprovado ou Reprovado. Ao final, o programa deverá exibir a quantidade de alunos Aprovados e a quantidade de Reprovados.

Exercício 3.2 - Considerando que o fatorial de um número é o resultado da multiplicação entre si dos algarismos a partir de 1 até o respectivo número. Ex: fatorial de 5 = 5 x 4 x 3 x 2 x 1. Fazer um programa em Java que receba a digitação de um número inteiro, calcule e exiba na tela o fatorial deste um número.

Exercício 3.3 - Fazer um programa em Java para receber via teclado os tipos de vinho existentes no estoque de uma vinícula. Considere que a vinícula possui apenas 3 tipos de vinho das seguintes uvas: Cabernet Sauvignon, Malbec e Tanat. Na digitação, o usuário irá digitar as seguintes letras para representar cada tipo de vinho:

- C para Cabernet Savion;
- M para Malbec;
- T para Tanat

Ao final, o programa deverá exibir na tela:

- A quantidade total de vinhos em estoque;
- A quantidade de cada vinho no estoque;

Obs: para indicar o final da digitação, o usuário vai digitar o tipo de vinho F (flag).

Exercício 3.4 - Fazer um programa em Java para receber via teclado a Matrícula (número), o Nome e as três Notas que os Alunos obtiveram na disciplina LTP durante o semestre. Considere que a Nota Final da disciplina é a soma destas três notas, e que serão aprovados os alunos com Nota Final maior ou igual a 60. Desta forma, para cada aluno digitado, exibir na tela a Nota Final na disciplina e se o aluno foi Aprovado ou Reprovado na disciplina. Ao final, exibir na tela:

- a quantidade total de alunos da turma;
- a média aritmética de notas finais da turma;
- o Nome e a nota do aluno que obteve a maior nota final;

Obs: para indicar o final da digitação, o usuário vai digitar a Matrícula 999 (flag).

6.3 – Estrutura de repetição *do... while* (repita...até)

Se caracteriza por executar a lista de comandos contidos no loop antes de verificar a condição que vem no final. Após executar os comandos do loop, é verificada a condição. Enquanto a condição for verdadeira, os comandos que estão dentro da estrutura de repetição continuam sendo executados. Se a condição for falsa, a repetição é encerrada e o programa segue executando os demais comandos após o final da repetição, ou seja, fora do loop.

Sintaxe:

Um comando dentro da repetição	Mais de um comando dentro da repetição

`do {`
`  comando1;`
`} while ( condição );`
	`do {`
	`  comando1;`
	`  comando2;`
	`} while ( condição );`

Obs: quando existir apenas uma linha de comando dentro da estrutura de repetição `do...while`, não é necessário que este comandos esteja entre chaves - `{ }`, no entanto, é pelos padrões de codificação da Sun, as chaves devem ser utilizadas sempre.

Exemplo 1: Dada a altura e a largura de um retângulo, calcular e imprimir a área. Em seguida, por 5 vezes incrementar 2 na altura e decrementar 1 na largura, e em cada uma das vezes calcular e imprimir novamente a área do retângulo.

```
int area;
int altura = 2;
int largura = 6;
int x = 1;
do{
  area = altura * largura;
  System.out.println("A Área é: " + area );
  altura = altura + 2;
  largura --;
  x ++;
} while (x <= 5);
```

No exemplo acima, a execução da lógica de repetição acontecerá exatamente da mesma forma que no Exemplo 3 da capítulo 5.1 (while). Sendo assim, os resultados são exatamente os mesmos.

Obs: O comado ***do...while*** é muito similar ao ***while***, se diferencia somente por testar a condição apenas após a execução da lista de comandos que estiver escrita dentro da estrutura de repetição. Sendo assim, ele deve ser utilizado em situações onde o programador deseja garantir que os comandos existentes da estrutura de repetição serão ser executados pelo menos uma vez, independentemente se a condição do loop é verdadeira ou falsa desde o início.

6.4 – Estrutura de repetição *for* (para...faça).

Caracteriza-se por repetir um conjunto de comandos uma quantidade de vezes pré-definida. O loop é encerrado quando as repetições já tiverem ocorrido a quantidade de vezes pré-determinada, ou seja, quando a condição assumir o valor falso.

Sintaxe:

```
for (contador = valor inicial; condição ;incremento ou decremento) {
  comando1;
}

for (contador = valor inicial; condição ; incremento ou decremento) {
  comando1;
  comando2;
}
```

Obs: quando existir apenas uma linha de comando dentro da estrutura de repetição `for`, como é o caso da primeira opção de sintaxe acima, não é necessário que este comandos esteja entre chaves - `{ }`, no entanto, é pelos padrões de codificação Java da Sun Microsystem, as chaves devem ser utilizadas sempre.

→ O `contador` deverá ser uma variável de tipo inteiro que será utilizada para controlar o número de repetições.
→ O `valor_inicial` é o valor inicial que o `contador` será inicializado antes da primeira execução dos comandos.
→ A `condição` deverá ser criada de forma a indicar um valor limite que o `contador` assumirá, e quando a condição se tornar falsa, ocorrerá o fim do comando `for`.
→ Se a `condição` criada for falsa desde o início, as linhas de comando escritas dentro do `for` não serão executadas nenhuma vez.
→ O comando de `incremento/decremento` indica qual será a alteração de valor do `contador` após cada rodada de execução dos comandos que estão dentro do `for`.
→ O `contador` poderá ser declarado na própria estrutura do `for` ou ser declarado em linhas de comando anteriores ao `for` , e ser apenas inicializado no `for;`
→ Caso o `contador` seja inicializado no próprio `for`, este `contador` não existirá mais na memória após o comando `for` ser finalizado. Ou seja o `contador` será neste caso uma variável local ao `for`, com valor na memória apenas enquanto a estrutura de repetição `for` estiver em execução.
→ Caso o `contador` seja inicializado antes do `for`, a variável `contador` continuará na memória após o comando `for` ser finalizado, e terá como conteúdo o ultimo valor assumido antes do término da execução do `for`.

Exemplo 1: Dada a altura e a largura de um retângulo, calcular e imprimir a área. Em seguida, por 5 vezes incrementar 2 na altura e decrementar 1 na largura, e em cada uma das vezes calcular e imprimir novamente a área do retângulo.

```
int area;
int altura = 2;
int largura = 6;
for (int x = 1; x <= 5; x++) {
  area = altura * largura;
  System.out.println("A Área é: " + area );
  altura = altura + 2;
  largura --;
}
```

Obs: neste exemplo, a variável `X` foi declarada dentro da estrutura do `for`, desta forma, a mesma não existirá mais na memória após o fecha-chaves `}` da estrutura for.

Exemplo 2:

```
byte x;
for (x = 1; x <= 9; x = x + 2) {// x será incrementado de 2 em 2
  System.out.println("O valor de X é: " + x );
}
System.out.println("Valor de x: " + x ); // qual o valor de X aqui ?
```

Obs: neste exemplo, a variável `X` foi declarada antes do início da estrutura `for`, desta forma, a mesma continuará existindo na memória após a linha fecha-chaves `}` da estrutura for.

Exemplo 3:

Para se executar o ***for*** com passo negativo, ou seja, ao invés do valor do contador ser incrementado a cada rodada de execução do for, ele ser decrementado de 1 ou outro valor.

```
int area;
int altura = 2;
int largura = 6;
for (int x = 5; x >= 1; x--) {     // x será decrementado de 1 em 1
  area = altura * largura;
  System.out.println("A Área é: " + area );
  altura = altura + 2;
  largura --;
}
```

6.5 - Comparação entre o *while* , *do...while* e *for*

O ***while*** deve ser utilizado em situações onde os comandos do loop só devem ser executados se a condição do loop for verdadeira antes de iniciar o loop.

O ***do...while*** deve ser utilizado em situações onde os comandos do loop devem ser executados pelo menos uma vez independente se a condição do loop é verdadeira ou falsa.

O ***for*** deve ser utilizado em loops onde o número de repetições é previamente conhecido e não há condições para que estas repetições ocorram, elas simplesmente devem ocorrer em uma quantidade de vezes pré-determinada .

O quadro abaixo contém uma mesma lógica escrita com cada um dos três comandos de loop estudados. Desta forma podemos comparar as diferenças de sintaxe entre eles:

```
int area;
int altura = 2;
int largura = 6;
int x = 1;
while (x <= 5) {
  area = altura * largura;
  System.out.print(area);
  altura = altura + 2;
  largura --;
  x ++;
}
```

```
int area;
int altura = 2;
int largura = 6;
int x = 1;
do {
  area = altura * largura;
  System.out.print(area);
  altura = altura + 2;
  largura --;
  x ++;
} while (x <= 5);
```

```
int area;
int altura = 2;
int largura = 6;
for (int x=1; x<=5; x++) {
  area = altura * largura;
  System.out.print(area);
  altura = altura + 2;
  largura --;
}
```

6.6 - O comando `break` utilizado para interromper qualquer estrutura de repetição

O comando `break` interrompe qualquer uma das estruturas de repetição (while, do...while ou for), fazendo com que, após a interrupção, a execução do programa prossiga a partir da primeira linha após o o fecha-chaves `}` que finaliza comando de repetição.

Exemplo 1:

```
int idade;
char sexo;
int x;
int quantEmpregados = 0;
for (x = 1; x <= 5; x++) {
  System.out.print("Digite a idade do empregado: ");
  idade = leia.nextInt();
  if (idade == 0) {
    break;
  }
  System.out.print("Digite o sexo do empregado: ");
  sexo = leia.next().charAt(0);
  quantEmpregados ++;
}
System.out.print("A quantidade de empregados é: " + quantEmpregados);
```

Neste exemplo acima, apesar do comando `for` utilizar o contador `x` que varia de `1` a `5`, se o usuário digitar na variável `idade` o valor `0` (zero), o `for` será finalizado pelo comando `break`, e a próxima linha de comando a ser executada será a que existir logo após o fecha-chaves `}` que finaliza o `for`, que é a exibição na tela da quantidade de empregados contida na variável `quantEmpregados;`

6.7 – Uso de Estrutura de Repetição para Consistência (ou validação) da Entrada de Dados.

Quando um programa solicita ao usuário que digite um dado, o mesmo digitará um valor conforme sua escolha e desejo. Entretanto, a maior parte dos dados possuem uma faixa limitada de valores válidos ou aceitáveis. Como por exemplo a idade de uma pessoa. Nenhuma pessoa poderia ter idade negativa, assim como também não seria possível uma pessoa ter 200 anos de idade. Desta forma, para evitar que usuários digitem na entrada de dados valores impossíveis ou inválidos, o programa deverá validar, ou consistir, a digitação.

A consistência na entrada de dados é uma forma de não aceitar valores impossíveis ou inválidos digitados pelo usuário. Desta forma, o programa deverá testar o valor digitado, e caso seja inválido, deve informar ao usuário e solicitar uma nova digitação

Para criar esta consistência, utiliza-se uma estrutura de repetição do tipo:

```
do{
  <entrada de dados>;
} while (<valor digitado> != <valor válido>);
```

Ou seja, a estrutura de repetição vai garantir que o programa continuará solicitando ao usuário a digitação do dado enquanto o valor digitado for inválido.

Exemplo de lógica de validação com o comando `do...While`:

```
int idade;
Scanner leia = new Scanner(System.in);
do{
   System.out.println("Digite a idade do aluno: ");
   idade = leia.nextInt();
   if (idade < 0 || idade > 150) {
      System.out.print("! Digite idade entre 0 e 150");
   }
} while (idade < 0 || idade > 150);
```

Nesta lógica, o usuário será repetidamente solicitado que digite a idade do aluno enquanto o valor digitado for menor que 0 ou maior que 150.

O comando `do..while` é o que melhor se encaixa na lógica de repetição para a consistência já que só testa a condição após ter executado todo o conjuto de comando pelo menos uma vez. Entretanto, a lógica também pode ser feita com o comando `while`:

```
int idade = 151;
Scanner leia = new Scanner(System.in);
while (idade < 0 || idade > 150)
   System.out.println("Digite a idade do aluno: ");
   idade = leia.nextInt();
   if (idade < 0 || idade > 150) {
      System.out.print("! Digite idade entre 0 e 150");
   }
};
```

A desvantagem do `while` é a necessidade em inicializar a variável que receberá a digitação com um valor inválido, caso contrário a condição do `while` já será falsa desde o início e não seriam executados os comandos escritos entre o `while` e o `fecha-chaves }`.

6.8 – Comparar se duas Strings são iguais

```
STRING_1.equals(STRING_2)
```

O método `equals`, compara duas strings e retorna `true` se o conteúdo das duas for totalmente igual. Caso contrário, o método retorna `false`.

OBS: letras maiúsculas são diferentes de minúsculas.

Exemplo 1:

```
String texto1 = "DINAMICA"
String texto2 = "dinamica";
if (texto1.equals(texto2)) {  // neste caso texto1 não é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

Exemplo 2:

```
String texto1 = "Dinamica";
String texto2 = "dinamica";
if (texto1.equals(texto2)) {// novamente texto1 não é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

Exemplo 3:

```
String texto1 = "dinamica";
String texto2 = "dinamica";
if (texto1.equals(texto2)) {// agora sim, texto1 é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

EXERCÍCIOS (parte 4):

Exercício 4.1- Fazer um programa em Java para receber via teclado o NOME, o SALÁRIO e o NÚMERO DE DEPENDENTES dos 10 empregados de uma empresa. O programa deverá calcular um Novo Salário para cada empregado de acordo com a tabela abaixo:

- salários abaixo de R$1.000,00 terão aumento de 30%
- salários de R$1.000,00 até R$2.000,00 terão aumento de 20%
- salários acima de R$2.000,00 terão aumento de 10%

Em seguida, o programa deverá acrescentar ao Novo Salário calculado R$50,00 por DEPENDENTE e imprimir o Novo Salário.

Ao final o programa deverá imprimir:
- Soma dos Novos Salários
- Média dos Novos Salários
- Número de empregados que passou a receber salário acima de R$1700,00.

Exercício 4.2 - Existem 3 candidatos à prefeito para a cidade XYZ. Feita a eleição, os votos serão digitados contendo os seguintes valores:

1 → indica que o voto foi no candidato número 1
2 → indica que o voto foi no candidato número 2
3 → indica que o voto foi no candidato número 3
4 → indica que o voto foi nulo
5 → indica que o voto foi em branco

Sabendo que 150 eleitores compareceram a eleição e depositaram seus votos na urna, fazer um programa em Java que receba via teclado a digitação dos votos contidos na urna. Admitindo-se que não ocorrerão empates entre os candidatos e que a eleição seja válida, ou seja, um dos 3 candidatos será o vencedor, o algoritmo deverá calcular e imprimir :

- qual foi o candidato vencedor;
- o número de votos em branco;
- o número de votos nulos;

obs.: - consistir a digitação dos votos e aceitar apenas os números de 1 a 5.
.

Exercício 4.3 – Fazer um programa em Java para receber via teclado a ALTURA em metros e o SEXO dos atletas de uma delegação. O programa deverá calcular e imprimir:
- a maior altura encontrada
- a menor altura encontrada
- o número de atletas do sexo feminino
- a média da altura masculina
- a média geral das alturas.

Obs:
- adotar um flag para encerrar a entrada de dados;
- consistir os valores digitados na entrada de dados de maneira que só poderão ser aceitos:
  SEXO = M ou F;
  ALTURA maior que zero e menor ou igual a 2,5 metros;

Exercício 4.4 - Em uma Fábrica trabalham empregados divididos em 3 classes: A, B e C. Fazer um programa em Java que receba via teclado o CÓDIGO DO OPERÁRIO (inteiro), a CLASSE do operário (caracter 1 posição) e o NÚMERO DE PEÇAS FABRICADAS no mês (inteiro). Em seguida o programa deverá calcular e imprimir o salário dos empregados considerando a tabela a seguir:

CLASSE	Peças fabricadas no mês	Calculo do Salário
A	até 30	R$500,00 + R$2,00 por peça fabricada
A	de 31 a 40	R$500,00 + R$2,30 por peça fabricada
A	acima de 40	R$500,00 + R$2,80 por peça fabricada
B	-	R$1.200,00
C	Até 50	R$ 40,00 por peça fabricada
C	acima de 50	R$ 45,00 por peça fabricada

Ao final, o programa deverá imprimir:

- Total gasto pela empresa com o pagamento de salários
- Número total de peças fabricadas
- Código do Operário que fabricou o menor número de peças
- Média de quantidade de peças fabricadas por empregados da classe B

Obs:

- adotar o flag Código de Operário = 0 (zero) para encerrar a entrada de dados;
- consistir os seguintes valores digitados na entrada de dados:
  - NÚMERO DE PEÇAS deverá ser maior que zero
  - CLASSE deverá ser A, B ou C

Exercício 4.5 - Pedra-papel-tesoura é um jogo envolvendo dois jogadores no qual cada jogador escolhe uma de 3 jogadas possíveis: “pedra”, “papel” ou “tesoura”. O resultado do jogo é determinado com base nas seguintes regras:

- O jogador que escolher “pedra” perde o jogo se o outro escolher “papel” e ganha caso o outro escolha “tesoura”.
- O jogador que escolher “papel” perde o jogo se o outro escolher “tesoura” e ganha caso o outro escolha “pedra”.
- O jogador que escolher “tesoura” perde o jogo se o outro escolher “pedra” e ganha caso o outro escolha “papel”

Caso ambos escolham a mesma jogada, o jogo é considerado um empate.

Fazer um programa para identificar e exibir os ganhadores de um jogo pedra-papel-tesoura. O programa deverá receber como entrada de dados a digitação de sucessivas jogadas escolhida pelo jogador 1 e pelo jogador 2. Após cada jogada, o programa deverá identificar e exibir na console o jogador vencedor.

O programa deve terminar se um dos jogadores introduzir uma jogada inválida.

Seguem alguns exemplos:

```
Digite a jogada do Jogador 1: pedra
Digite a jogada do Jogador 2: papel

Jogador 2 venceu o jogo!

Digite a jogada do Jogador 1: tesoura
Digite a jogada do Jogador 2: papel
```

```
Jogador 1 venceu o jogo!
Digite a jogada do Jogador 1: tesoura
Digite a jogada do Jogador 2: tesoura

Empate!

Digite a jogada do Jogador 1: xyz
Digite a jogada do Jogador 2: tesoura

Jogo terminado.
```

Exercício 4.6 - Uma empresa aérea quer controlar os custos de seus voos. Ela possui três voos diários de números 1,2 e 3. Todos os aviões possuem 100 lugares e os voos possuem preços diferenciados de passagem:

→ voo 1 ..... passagem R$ 100,00
→ voo 2 ..... passagem R$ 150,00
→ voo 3 ..... passagem R$ 200,00

Porém o preço das passagens pode cair de acordo com a lotação de cada voo :

→ voo com 70 passageiros ou mais ..... passagem com 60% de desconto
→ voo com 50 a 69 passageiros ..... passagem com 30% de desconto
→ voo com menos de 50 passageiros ..... passagem sem desconto

Faça um programa em Java que receba via teclado a digitação da venda de passagens para os 3 voos contendo cada uma o Número do voo e a Quantidade de passagens vendidas. O algoritmo deverá calcular e imprimir o preço de cada passagem baseado na lotação do voo.

Como resultados finais, o algoritmo deverá imprimir :

a) o número do voo que arrecadou maior valor e o valor arrecadado
b) a média da arrecadação dos 3 voos
c) o valor da passagem mais barata e o número do voo correspondente

obs. :
→ consistir a digitação do número do voo de forma a aceitar somente os valores 1, 2 ou 3
→ consistir a digitação do número de passagens vendidas para aceitar valores entre 0 e 100.

7 - Estruturas Homogêneas de dados

O que seria uma estrutura homogênea de dados ?

É uma estrutura capaz de armazenar um conjunto de dados homogêneos, ou seja, dados do mesmo tipo.

Por exemplo:

- os nomes de 20 pessoas
- os salários de 200 empregados
- as datas de nascimento de 40 pessoas
- e outros...

Existem dois tipo de estruturas homogêneas de dados, os **vetores** e as **matrizes**. A diferença entre elas é que o vetor é uma estrutura unidimensional, ou seja, armazena dados em apenas uma dimensão. Já as matrizes pode ser bidimensionais ou tridimensionais.

Se um conjunto de dados for heterogêneo não poderia, ou não deveria, ser armazenado em um vetor ou uma matriz.

Exemplo de conjunto de dados heterogêneos:

- nome, endereço, idade e sexo de um aluno;
- nome, cargo, salário e formação de um empregado

Um vetor ou matriz não poderia armazenar estes dados porque são tipos diferentes de dados. Entretanto, um vetor ou matriz poderia armazenar por exemplo um conjunto só de nomes, ou um conjunto só de endereços, ou um conjunto só de idades de alunos.

Desta forma, sempre que for necessário armazenar um conjunto de dados homogêneos, ao invés de criar um conjunto de variáveis (uma para armazenar cada valor), deverá ser criado um único vetor ou matriz que armazenará todos os dados.

Em variáveis de memória só é possível armazenar um único valor por vez. Considere por exemplo um programa que precisa armazenar as notas de 5 provas realizadas por um aluno. Uma opção para isto seria criar cinco variáveis para armazenar as notas, uma para cada nota. Por exemplo:

```
int nota1, nota2, nota3, nota4, nota5;
```

No entanto, esta não é a melhor maneira para trabalhar estes dados pois como se tratam de 5 variáveis diferentes, com nomes diferentes, necessitariam também 5 comandos diferentes de entrada de dados para recebê-los e também 5 estruturas de cálculo para manipulá-los. Sendo assim, a melhor opção seria criar um único vetor contendo 5 posições de armazenamento, uma para cada valor.

7.1 – Vetores

Este tipo de estrutura é também chamado de **matriz unidimensional**. Um vetoré declarado com seu nome, tamanho e tipo. Para declarar um vetor em java basta abrir e fechar um colchete antes ou após a declaração de uma variável, desta forma esta variável na verdade será um vetor. Em seguida, deverá ser informado o tamanho do vetor. Em java a primeira posição de um vetor é sempre a de número 0 (zero) e a ultima posição será o tamanho do vetor menos 1.

7.1.1 – Declaração de vetores.

Exemplo 1:

```
float notas[];            // definição da notas como um vetor
notas = new float[10];    // definição do tamanho do vetor notas com 10 posições
```

No exemplo acima:

→ `notas` é o nome do vetor;
→ `new float[10]` indica que o vetor armazenará até 10 notas diferentes do tipo **`float`**, e cada nota terá um índice de localização dentro do vetor que inicia em 0 e termina em 9;

Imagine este vetor acima como um conjunto de caixinhas numeradas de 0 a 9, cada uma delas armazenará uma nota, ou seja, o vetor `notas` começa na posição 0 e termina na posição 9, total de 10 itens:

Posições do vetor →	0	1	2	3	4	5	6	7	8	9
notas										

Obs: Uma variável somente pode conter um valor por vez. No caso dos vetores, estes poderão armazenar mais de um valor por vez, pois são dimensionados exatamente para este fim.

Exemplo 2:

A declaração e a inicialização de um vetor também pode ser escrita em apenas uma linha de comando:

```
String nomeAlunos[] = new String[6];
```

Imagine o vetor acima como um conjunto de caixinhas numeradas de 0 a 6, cada uma delas armazenará um nome de um aluno:

Posições do vetor →	0	1	2	3	4	5
nomeAlunos	JOSÉ	MÁRCIA	PEDRO	SILVIA	SANDRA	JORGE

Neste exemplo o nome JOSE foi armazenado no vetor na posição 0, o nome MARCIA foi armazenado na posição 1, o nome PEDRO foi armazenado na posição 2, e assim em diante.

7.1.2 - Atribuindo valores ao vetor por meio de comando de atribuição.

Exemplo 1:

```
float notas[] = new float[5];
```

A atribuição ou a referencia a qualquer dos valores contidos no vetor é feita indicando a posição do vetor entre colchetes. Exemplo: Atribuindo valores ao vetor de notas:

```
notas[0] = 5.2;
notas[1] = 8.0;
notas[2] = 9.2;
notas[3] = 7.5;
notas[4] = 8.3;
```

Em cada linha de comando o nome do vetor é o mesmo, o que muda é a informação indicada dentro dos colchetes, ou seja, o índice ou posição do vetor. O índice representa o local dentro do vetor onde o valor será armazenado. Para o exemplo anterior teríamos:

notas

0	1	2	3	4	=> índice ou posição no vetor
5.2	**8.0**	**9.2**	**7.5**	**8.3**	=> conteúdo de cada posição

7.1.3 – Preencher um vetor com valores digitados por um usuário (Entrada de Dados).

Exemplo 1:

```
import java.util.*;
public class EntradaDadosVetor1 {
public static void main(String[] args) {
    // 1 - Declaração e inicialização de variáveis
    int contador;
    float notasAlunos[] = new float[15];
    Scanner leia = new Scanner(System.in);
    // 2 - Entrada de dados
    for (contador = 0; contador <= 14; contador++) {
      System.out.print("Digite a nota " + contador + ": ");
      notaAlunos[contador] = leia.nextFloat();
    }
}
```

Neste exemplo a variável `contador` que controla o número de repetições do `for` será usada para como índice do vetor onde o dado será armazenado. O `contador` é inicializado com o valor 0 (zero) no primeiro ciclo do comando `for`. Desta forma, o primeiro valor digitado pelo usuário será armazenado na posição 0 (zero) do vetor. No segundo ciclo do `for`, o `contador` será acrescido de 1, então passará a valer 1. Então, a próxima nota digitada será armazenada na posição 1 do vetor. E assim sucessivamente.

Exemplo 2:

Imagine uma situação onde você precisa receber e armazenar a digirtação do nome e a nota dos 10 alunos de uma turma. Neste caso, ao invés de criar 10 variáveis para armazenar os nomes e outras 10 para armazenar as notas, vamos criar dois vetores com 10 posições cada um. Um para armazenar as notas e outro para armazenar os nomes. Após a digitação dos 10 nomes e 10 notas, os vetores serão preenchidos e ficarão como no exemplo abaixo:

	nomes	notas
0	Marcelo	8
1	Sandra	2
2	Regina	9
3	Pedro	5
4	Beatriz	3
5	Alberto	7
6	Juarez	9
7	Márcia	10
8	Fernanda	5
9	Cesar	10

O código abaixo é um exemplo que poderia ser usado para preencher os vetores:

```
import java.util.*;
public class EntradaDadosVetor2 {
public static void main(String[] args) {
    // 1 - Declaração e inicialização de variáveis
    int x;  // contador
    String nomes[] = new String[10];
    float notas[] = new float[10];
    Scanner leia = new Scanner(System.in);
    // 2 - Entrada de dados
    for (x = 0; x <= 9; x++) {
      System.out.print("Digite o nome " + x + ": ");
      nomes[x] = leia.nextLine();
      System.out.print("Digite a nota " + x + ": ");
      notas[x] = leia.nextFloat();
    }
}
```

No exemplo acima, a variável `x` foi inicializada com o valor 0 (zero) no primeiro ciclo do `for`. Desta forma, o primeiro nome digitado será armazenada na posição 0 (zero) do vetor `nomes`, e a primeira nota digitada na posição 0 do vetor `notas`. No segundo ciclo do `for` o `x` passará a valer 1, e assim por diante.

7.1.4 - Exercício resolvido:

Faça um programa em Java que receba via teclado as notas dos 40 alunos de uma turma. O programa deverá calcular e imprimir a média de notas da turma e o número de alunos que tiraram nota acima da média da turma.

Neste caso, primeiramente deveremos receber as 40 notas para depois calcular a média da turma. Em seguida, poderemos comparar cada nota com a média e contar quantas são acima da média. Para fazer esta comparação as 40 notas deverão ser armazenadas em memória.

Para armazenar as 40 notas na memória, uma opção seria criar 40 variáveis, uma para cada nota. Mas esta não seria a forma mais correta, pois que teríamos que escrever várias linhas de código para receber cada nota e depois comparar com a média.

Sendo assim, a forma correta para armazenar as 40 notas é criar um único vetor com 40 posições.

Planejamento do código:

1 – Declaração de variáveis:
    notas (40), contador, somaNota, mediaNota, quantAcimaMédia
2 – Entrada de dados:
    - receber as 40 notas
3 – Cálculos:
    - somar as notas
    - calcular a média das notas
    - calcular quantas notas ficaram acima da média
4 – Saída de dados:
    - imprimir a média e a quantidade de notas acima da média

```java
import java.util.*;
public class ExemploVetores1 {

   public static void main(String[] args) {
      // 1 - Declaração de variáveis
      int notas[] = new int[40];
      byte contador;
      int somaNotas = 0;
      float mediaNotas;
      byte quantAcimaMedia = 0;
      Scanner leia = new Scanner(System.in);

      // 2 - Entrada de dados
      for (contador = 0 ; contador <= 39 ; contador++) {
         System.out.print("Digite a nota do aluno " + contador + 1 + ": ");
         notas[contador] = leia.nextInt();

         // 3 - cálculos
         somaNotas = somaNotas + notas[contador];
      }
      mediaNotas = somaNotas / 40;
      for (contador = 0 ; contador <= 39 ; contador++)
         if (notas[contador] > mediaNotas) {
             quantAcimaMedia ++;
         }
      }
      // 4 - saída de dados
      System.out.println("Media das notas: " + mediaNotas);
      System.out.println("Quantidade de alunos com nota acima da media: " +
            quantAcimaMedia);

  }
}
```

EXERCÍCIOS (parte 5):

Exercício 5.1 – A tabela a seguir contém em cada linha as 5 notas de provas obtidas por 30 alunos durante o período. O índice das linhas corresponde ao número do aluno, e o das colunas corresponde ao número da nota:

	1	2	3	4	5
1	6.0	7.0	6.5	8.0	4.0
2	9.0	8.0	4.0	9.0	6.0
3	5.5	7.5	4.5	8.5	9.0
...	...	...	...	...	...
30	8.0	9.0	8.5	7.5	9.0

Fazer um programa em Java para receber via teclado as 5 notas de cada um dos 30 alunos. O programa deverá calcular a média de cada aluno e a média da turma. Como resultado final o algoritmo deverá imprimir o relatório a seguir:

número do aluno	média do aluno
1	6.3
2	7.8
3	7.0
...	...
30	8.4

média da turma: 7.38

Exercício 5.2 - Faça um programa em Java que receba via teclado um conjunto de 100 notas dos alunos da disciplina Linguagem e Técnicas de Programação. O programa deverá calcular e imprimir uma tabela contendo os valores das notas e a frequência absoluta da nota.

Observações:
- as notas digitadas serão inteiras de 0 a 10;
- frequência absoluta é o número de vezes que uma nota aparece no conjunto.

Exemplo do relatório:

Tabela de notas e frequência

Nota	frequência absoluta
0	5
1	8
2	3
...	...
9	23
10	13

Exercicio 5.3 - A Fumec deseja saber se existem alunos cursando simultaneamente as disciplinas Linguagem e Técnicas de Programação (LTP) e Cálculo. Faça um programa em Java que receba via teclado o número de matrícula dos alunos que estão cursando LTP (máximo 150 alunos) e dos alunos que estão cursando Cálculo (máximo 220). O algoritmo deverá calcular e imprimir o número de alunos que estão matriculados nas duas disciplinas.

Obs: - Tanto para a digitação das matrículas na disciplina LTP quanto na disciplina Cálculo, adotar a matrícula 999 como *flag* para encerrar a entrada e dados.

Exercício 5.4 - Faça um programa em Java para corrigir as provas de uma disciplina. A prova é composta de 10 questões de múltipla escolha, cada uma com alternativas de A até E. Primeiramente o programa deverá receber via teclado a digitação do gabarito de respostas da prova. em seguida, o programa deverá receber para cada aluno a digitação do número de matrícula e as respostas das 10 questões da prova. O programa deverá calcular e imprimir a nota de cada aluno.

Como resultados finais o programa deverá imprimir:
- o percentual de alunos aprovados (60% de acertos ou mais)
- a nota que apareceu com maior frequência na turma.

Observações:
- adotar um FLAG para encerrar a entrada de dados.
- para este exercício você deverá criar um vetor para armazenar o Gabarito e outro acumular a frequência de notas da turma (Similar ao exercício 5.2 desta atividade).
- para calcular o percentual aprovado use a fórmula:

Percentual = Num. Alunos Aprovados * 100 / Num. Alunos na Turma

Exercício 5.5 - Faça um programa em Java que receba via teclado o Nome, Nota e Sexo dos alunos da disciplina Linguagem e Técnicas de Programação. O programa deverá imprimir se o aluno está APROVADO ou REPROVADO, sabendo que a nota mínima para aprovação é 60.

Como resultados finais o programa deverá imprimir:
- o número de aprovações do sexo masculino;
- o número de reprovações do sexo feminino;
- a média das notas femininas;
- a média geral das notas;
- relatório somente dos alunos aprovados contendo o Nome e a Nota.

Consistências:
- aceitar somente notas inteiras entre 0 e 100;
- aceitar somente os valores M ou F para sexo;

Observações:
- a turma tem 50 alunos;
- você vai precisar criar um vetor para armazenar o nome dos aprovados e outro para armazenar a nota dos aprovados para imprimir o relatório no final.

Exemplo de relatório de alunos Aprovados:

RELATÓRIO DE APROVADOS

NOME	NOTA
MARCELO RIBEIRO	75
DENISE FARIA	61
MARIA DAS DORES	82
...	...

7.2 – Matrizes

A matriz é uma estrutura homogênea de dados com mais de uma dimensão. Na maioria das vezes as matrizes são bidimensionais, mas algumas linguagens de programação permitem 3 ou mais dimensões. O Java permite até 128 dimensões.

Em que situações devemos utilizar uma matriz para armazenar dados ?

Quando precisamos armazenar um conjunto de dados homogêneos , ou seja, do mesmo tipo, e existe a necessidade de guarda-los dentro da matriz em mais de uma dimensão.

7.2.1 – Declaração de matrizes.

Vamos supor que será necessário armazenar as notas de 4 disciplinas diferentes para 5 alunos. Uma opção seria criar 5 vetores, onde cada um armazenaria as 4 notas das disciplinas. Entretanto, esta não é a melhor solução já que seria necessário criar uma estrutura de controle para cada vetor. A melhor opção então seria criar uma única matriz de duas dimensões para armazenar as notas das 4 disciplinas para cada um dos 5 alunos.

Para declarar matrizes em java é muito similar a forma de declaração de vetores, diferenciando-se apenas no número de dimensões, já que o vetor possui apenas uma dimensão e matrizes possuem 2 ou mais dimensões. Para uma matriz de 2 dimensões a primeira dimensão indica o número de linhas e a segunda o número de colunas da matriz.

Para o caso de armazenar as notas de 4 disciplinas para os 5 alunos, a declaração da matriz seria:

```
float notas[][];
notas = new float[5][4];

ou

float notas[][]= new float[5][4];
```

Na declaração de matrizes em Java, o primeiro valor informado dentro dos colchetes (5) indica o número de linhas da matriz, e o segundo valor (4) o número de colunas.

Neste caso, as linhas representam os Alunos, e as colunas representas as Disciplinas:

Disciplinas

ALUNOS	0	1	2	3
0	**8.5**	**9.0**	**7.8**	**8.9**
1	**5.0**	**6.8**	**8.7**	**6.5**
2	**7.0**	**7.5**	**5.7**	**7.8**
3	**8.5**	**8.0**	**9.2**	**7.9**
4	**5.5**	**8.0**	**7.2**	**7.0**

Índice das colunas => 4 disciplinas

Índice das linhas => 5 alunos

Considerando as notas acima colocadas em cada posição da matriz, podemos concluir que:

- o Aluno da linha 1 tirou nota 8.7 na Disciplina da coluna 2;
- o Aluno da linha 3 tirou nota 8.0 na Disciplina da coluna 1;
- o Aluno da linha 0 tirou nota 8.9 na Disciplina da coluna 3;

7.2.2 - Atribuir um valor para uma posição da matriz

```
float notas[][]= new float[5][4];

notas[0][0] = 76;
notas[0][1] = 72;
notas[1][0] = 45;
notas[2][2] = 75;
notas[3][1] = 70;
notas[4][3] = 80;
```

	Disc 0	Disc 1	Disc 2	Disc 3
Aluno 0	76	72		
Aluno 1	45			
Aluno 2			75	
Aluno 3		70		
Aluno 4				80

O comando de atribuição = pode ser utilizado para atribuir um valor a uma posição de uma matriz, de forma similar a atribuição de valor a um vetor ou a uma variável.

A matriz bidimensional utiliza dois índices de controle, o primeiro para indicar a linha e segundo para indicar a coluna de localização do dado.

A posição de um dado na matriz sempre deverá ser informada, entre colchetes, contendo a linha e a coluna de localização do dado.

7.2.3 - Preencher uma matriz com valores digitados pelo usuário (Entrada de Dados)

Exemplo:

```
import java.util.*;
public class EntradaDadosMatriz1 {
public static void main(String[] args) {
   // 1 - Declaração e inicialização de variáveis
   byte linha;
   byte coluna;
   float notas[][] = new float[5][4];
   Scanner leia = new Scanner(System.in);
   // 2 - Entrada de dados
   for (linha = 0; linha <= 4; linha++) {
     for (coluna = 0; coluna <= 3; coluna++) {

       System.out.print("Digite a nota do aluno " + linha + " para a
            disciplina " + coluna + ": ");
       notas[linha][coluna] = leia.nextFloat();
     }
   }
```

No exemplo acima a variável `linha` controla o índice das linhas da matriz e a variável `coluna` controla o índice das colunas.

7.2.4 - Exercício Resolvido:

Faça um programa em Java que receba via teclado uma matriz 4 x 5. O algoritmo deverá somar e imprimir os elementos de cada linha e como resultado final imprimir a soma total dos elementos da matriz.

Neste caso, deveremos receber os dados da matriz linha a linha para ir calculando e imprimindo as somas por linha.

	0	1	2	3	4		soma
0	8	2	5	7	1		23
1	7	1	6	0	3		17
2	2	10	5	3	9		29
3	6	8	5	9	10		38
						Soma total =	107

Planejamento do código:

1 – Declaração de variáveis:

numeros(4x5), linha, coluna, somaLinha, somaTotal

2 – Entrada de dados:

- receber a matriz numeros linha por linha

3 – Cálculos:

- somar os números das linhas
- acumular as somas das linhas para ter a soma total

4 – Saída de dados:

- imprimir as somas por linha
- imprimir a soma total

```
import java.util.*;
public class ExemploMatriz1 {

   public static void main(String[] args) {

      // Declaração de variáveis
      int numeros[][] = new int[4][5];
      byte linha;
      byte coluna;
      int somaLinha = 0;
      int somaTotal = 0;
      Scanner leia = new Scanner(System.in);

      // Entrada de dados
      for (linha = 0 ; linha <= 3 ; linha++) {
         for (coluna = 0 ; coluna <= 4 ; coluna++) {
               System.out.print("Digite o numero a ser armazenado na linha "
                       + linha + " e coluna " + coluna + ": ");
               numeros[linha][coluna] = leia.nextInt();

               // cálculos
               somaLinha = somaLinha + numeros[linha][coluna];
         }
         System.out.println("Soma dos numeros da linha " + linha +  ": " +
                somaLinha);
         somaTotal = somaTotal + somaLinha;
         somaLinha = 0;  // zerar o contador de linhas para iniciar a soma da
                              proxima linha
      }

      // saída de dados
      System.out.println("Soma de todos os numeros da matriz : " +
            somaTotal);

   }
}
```

EXERCÍCIOS (parte 6):

Exercício 6.1 – Uma Cia Aérea deseja fazer o controle das reservas de passagens para os seus voos. Fazer um programa em Java para receber via teclado a Fileira e a Cadeira (local do assento) e o Nome do Passageiro de cada reserva. O programa deverá calcular e imprimir para cada reserva o preço da passagem de acordo com a tabela abaixo:

- 1a. Classe	-	fileiras 1 a 3	- valor da passagem R$900,00
- Classe Executiva	-	fileiras 4 a 8	- valor da passagem R$700,00
- Classe Comercial	-	fileiras de 9 a 20	- valor da passagem R$350,00

Como resultados finais o programa deverá imprimir:
- o mapa do voo indicando o nome do cliente no respectivo local do assento.
- o total faturado no voo
- o total faturado na 1a. Classe
- número de assentos vagos nas janelas (cadeiras 1 e 6)

Consistência: para o local do assento, aceitar somente digitação de Fileira entre 1 e 20, e Cadeira entre 1 e 6.

Obs: - o Assento é composto pelo número da Fileira e número da Cadeira. O avião possui 20 fileiras com 6 cadeiras em dada uma. Exemplo: Assento 15 / 5 significa → Fileira 15 e Cadeira 5;
- adote um *flag* para encerrar a entrada de dados;
- você vai precisar criar uma matriz para armazenar o nome do passageiro no respectivo assento;

Exemplo de mapa de voo que deverá ser impresso:

	1	2	3	4	5	6
1		MARCIA	PEDRO			JOSIAS
2	SERGIO			MARCELA	APARECIDA	DAVI
3	VANDA					
4				PATRÍCIA	PAULO	TEREZA
...	...	...	...	...	...	...
20	FABRÍCIO	SAMIR	JORGE	VALÉRIA	MARTA	TELMA

Exercício 6.2 - Faça um programa em Java para receber via teclado os palpites de um jogador da loteria esportiva. Em seguida o algoritmo deverá calcular o valor a ser pago pela aposta. Os palpites do jogador deverá ser recebido da forma abaixo:

Jogos	1	2	3	4	5	6	7	8	9	10	11	12	13
Time 1 vendedor	-	-	-	-	X	-	-	-	X	-	X	-	-
Empate	X	-	X	X	-	X	-	-	X	X	X	X	X
Time 2 vencedor	X	X	-	-	-	-	X	X	-	X	X	-	X

Considerar que em cada digitação o usuário irá digitar a letra X ou - , o X indica o palpite escolhido. Se o usuário digitar apenas um X na coluna, representará uma escolha simples, dois X representa escolha dupla e três X escolha tripla.

O valor a ser pago pela a aposta será calculado da seguinte forma: $VALOR = V \times 2^D \times 3^T$

onde: - V é uma constante que deverá ter seu valor declarado inicialmente, e se refere ao valor do jogo para escolhas simples;
- D representa o número de escolhas duplas;
- T representa o número de escolhas triplas;

Após imprimir o valor da aposta, imprimir todo o bilhete de aposta no formato apresentado acima.

Exercício 6.3 - A tabela a seguir contém a quantidade de 3 produtos que estão estocados nos 3 armazéns de uma companhia . É fornecido também o custo de cada um dos produtos armazenados.

Faça um programa em Java para receber via teclado a digitação do custo de cada produto e armazenar em um vetor. Em seguida, o programa deverá receber e a quantidade em estoque de cada produtos em cada armazém, e armazenar em uma matriz.

Ao final o programa deverá calcular e imprimir:

- qual armazem possui maior número de produtos armazenados;
- o custo do estoque de cada produto em cada armazém;
- o custo total do estoque dos 3 produtos em cada armazém;
- o custo total de todos produtos em todos armazéns;

	PRODUTO 1	***PRODUTO 2***	***PRODUTO 3***
Armazem 1	1200	3700	3737
Armazem 2	1400	4210	4224
Armazem 3	2000	2240	2444

Custo R$ / unidade	R$260,00	R$420,00	R$330,00

8 – Métodos da linguagem Java - Parte 1

8.1 – Comparar se duas Strings são iguais

```
STRING_1.equals(STRING_2)
```

O método `equals`, compara duas strings e retorna `true` se o conteúdo das duas for totalmente igual. Caso contrário, o método retorna `false`.

OBS: letras maiúsculas são diferentes de minúsculas.

Exemplo 1:

```
String texto1 = "DINAMICA"
String texto2 = "dinamica";
if (texto1.equals(texto2)) {  // neste caso texto1 não é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

Exemplo 2:

```
String texto1 = "Dinamica";
String texto2 = "dinamica";
if (texto1.equals(texto2)) {// novamente texto1 não é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

Exemplo 3:

```
String texto1 = "dinamica";
String texto2 = "dinamica";
if (texto1.equals(texto2)) {// agora sim, texto1 é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

8.2 – Comparar se duas Strings são iguais (sem maiúsculo/minúsculo)

```
STRING_1.equalsIgnoreCase(STRING_2)
```

O método `equalsIgnoreCase`, compara duas strings e retorna `true` se as duas forem iguais mesmo quando existe diferença de maiúsculo e minúsculo entre elas. Caso contrário, o método retorna `false`.

Exemplo 1:

```
String texto1 = "DINAMICA";
String texto2 = "dinamica";
if (texto1.equalsIgnoreCase(texto2)) {  // neste caso texto1 é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

Exemplo 2:

```
String texto1 = "JOSE MARIA";
String texto2 = "Jose  maria";
if (texto1.equalsIgnoreCase(texto2)) {   // neste caso texto1 não é igual a texto2
  System.out.print("as Strings são iguais");
}else{
  System.out.print("as Strings NÃO são iguais");
}
```

OBS: neste Exemplo 2, o conteúdo de `texto1` possui 1 caracter espaço entre `JOSE` e `MARIA`, já o conteúdo de `texto2` possui 2 caracteres espaço entre `jose` e `maria`, sendo assim, possuem conteúdo diferentes independente de maiúsculo e minúsculo.

8.3 - Pesquisa Sequencial em Vetores

Este é o tipo de pesquisa mais simples em vetores, no qual cada item do vetor é examinado por vez e comparado ao item que se está procurando, até ocorrer uma coincidência. Este método de pesquisa é lento, porém, é a melhor opção para os casos em que os elementos do vetor encontra-se desordenados.

Exemplo de Pesquisa Seqüencial:

Criar um programa que inicialmente receba a digitação de 5 nomes e armazene em um vetor. Em seguida o programa receberá via teclado a digitação aleatória de Nomes e pesquisará se estes Nomes estão contidos no vetor:

Algoritmo:

1- Criar uma repetição para receber a digitação dos 5 nomes para gravar no vetor

2- Criar uma repetição que receba a digitação aleatória de nomes enquanto o usuário assim desejar.

2.1- Criar uma outra repetição para pesquisar o nome digitado no vetor:

- Comparar o nome digitado com o primeiro elemento do vetor;

- se for igual, o elemento procurado foi encontrado,
- caso contrário, avança para o próximo elemento do vetor.

- Se não for encontrado nenhum nome no vetor igual ao nome digitado, informar que não existe o elemento pesquisado.

3- Encerrar a pesquisa quando desejado.

```
import java.util.*;
public class PesquisaSequencialEmVetor {
  public static void main(String[] args){
    String nomePesquisa;
    String nomes[] = new String[5];
    int x;
    boolean encontrou;
    Scanner leia = new Scanner(System.in);

    for (x = 0 ; x <= 4 ; x++){
      System.out.print("Digite o Nome " + X + " para armazenar no Vetor: ");
      nomes[x] = leia.nextLine();
    }

    do{
      System.out.print("Digite o Nome para pesquisa (FIM para encerrar): ");
      nomePesquisa = leia.nextLine();
      if ( nomePesquisa.equals("FIM") ) {
      break;
      }
      encontrou = false;
      for (x = 0 ; x <= 4 ; x++){
        if ( nomePesquisa.equals(nomes[x]) ){
          encontrou = true;
          break;
        }
      }
      if (encontrou) {
        System.out.println("Nome encontrado na posição " + x + " do vetor!");
      }else{
        System.out.println("O Nome digitado NÃO FOI ENCONTRADO NO VETOR !");
      }
    }while ( ! nomePesquisa.equals("FIM"));
  }
}
```

EXERCÍCIOS (parte 7):

Exercício 7.1 – Fazer um programa em Java para consultar informações sobre os vôos de uma empresa aérea. Primeiramente o programa deverá receber via teclado os Nomes e respectivas Distâncias (em km, até Belo Horizonte) de até 20 Cidades no mundo e armazenar em dois vetores. Em seguida, o programa deverá entrar no módulo de consultas e solicitar o Nome de uma Cidade. Para cada cidade digitada o programa deverá calcular e exibir os seguintes resultados:

- Preço da Passagem
- Tempo de Vôo em minutos
- Número de escalas no percurso

Considerações para os cálculos:

- A empresa aérea cobra R$0,25 por km percorrido no vôo
- O avião voa com uma velocidade de 800 km/h
- A empresa aérea faz uma escala a cada 3 horas de vôo.

Observações:

- Adotar um flag para finalizar a entrada de dados com os Nomes de Cidades e Distâncias
- Adotar um flag para finalizar as consultas
- Exibir mensagem de erro se o Nome da Cidade digitada na consulta não existir na lista de cidades digitadas inicialmente (Max. 20)
- Durante a digitação dos Nomes das Cidades e Distâncias, consistir:
  - O Nome da Cidade deverá ser de preenchimento obrigatório
  - A distância até Belo Horizonte deverá ser no mínimo 500 km.

9 – Desenvolvimento de sub-rotinas em Java (métodos)

Na medida em que o nível de complexidade dos programas e o tamanho vão aumentando, muitas vezes escrevemos um mesmo conjunto de instruções em dois ou mais pontos do programa, ou em programas diferentes. O ideal seria se aquele conjunto de instruções pudesse ser escrito somente uma vez e utilizado quantas vezes fossem necessários.

Para isto, existem as sub-rotinas (chamadas de métodos em Java), ou seja, para facilitar e agilizar a criação de programas pelos programadores.

A idéia e dividir o programa em partes, e aquelas partes que necessitam ser usadas mais de uma vez sejam transformadas em sub-rotinas, evitando-se assim escrever um mesmo conjunto de comandos mais de uma vez. Ou seja, a seqüência de comandos é escrita somente uma vez, dentro da sub-rotina, e usada quantas vezes forem necessárias dentro de um mesmo programa ou em programas diferentes.

Outra vantagem gerada pela transformação de uma sequencia de comandos em sub-rotina, é reduzir o trabalho do programador quando este necessitar alterar algum comando da sub-rotina. Se o conjunto de comandos estiver escrito repetidas vezes dentro do mesmo programa ou em programas diferentes, o programador terá que alterar todas elas, tomando-se muito tempo e trabalho. Entretanto, se a lógica estiver escrita uma única vez em uma sub-rotina só precisará será alterada uma vez, independente se a sub-rotina estiver sendo usada uma ou mais vezes em um único programa ou em programas diferentes.

Uma terceira vantagem das sub-rotinas é a possibilidade de uso das mesmas por diferentes programadores por meio das bibliotecas de sub-rotinas. Isto evita que um programador tenha que desenvolver uma lógica que já foi criada por outro programador.

9.1 - Variáveis Globais e Locais

Uma variável é considerada *Global* quando é declarada dentro da classe mas fora de qualquer método, inclusive do método `main`. Uma variável *Global* poderá ser utilizada dentro de qualquer método existente na classe. Para a variável global ser utilizada em qualquer método ela deve ser declarada como `static`. (ver exemplo no item 14.3)

Uma variável é considerada *Local* quando é declarada dentro de um método e só é válida naquele método onde foi criada. Desta forma, se uma variável é declarada dentro de um método, os outros métodos contidos na classe não poderão fazer uso daquela variável, pois não visualizam a existência da mesma.

9.2 - Características dos Métodos

Os métodos possuem características que são apresentadas na declaração por modificadores como `public`, `static` e `void`.

Métodos do tipo `public` indica que poderá ser chamado por qualquer classe, `static` indica que o método é da classe, ou seja, poderá ser executado mesmo se nenhum objeto (instância) da Classe foi criado, `void` indica que o método não retornará valor após ter sido executado.

9.3 - Exemplo de um método que não retorna valor (`void`)

Um método `void` seráexecutado simplesmente quando o nome dele, seguido de um lista opcional de parâmetros, for escrito em uma linha de comando. Quando um método é chamado em uma linha

decomando,o controle de execução do programa é automaticamente transferido para o início do método. Após todos os comandos do métodos serem executados, ocontrole retorna automaticamente para linha de comando de onde ele foi chamado e em seguida será executada a próxima linha de comando.

```
import java.util.*;
public class Exemplo1{

  static int x,y;     // variáveis globais para todas as classes de Exemplo1

  public static void main(String Args[]){
    x = 5;
    y = 10;
    subtrair();
    System.out.println("Valor de X:" + x);       // X => 3
    System.out.println("Valor de Y:" + y);       // Y => 9
  }

  public static void subtrair(){
    x = x - 2;
    y = y - 1;
  }
}
```

9.4 - Parâmetros

Os parâmetros têm por finalidade servir como um ponto de comunicação entre o método e o programa de onde ele foi chamado. Desta forma,é possível passar valores de um programa para um método por meio de parâmetros. Dentro do método os parâmetros funcionam como uma variáveis Locais, ou seja, só podem ser utilizados dentro do próprio método.

Exemplo 2:

```
import java.util.*;
public class Exemplo2 {

  public static void main(String Args[]) {
    int x = 5;
    int y = 10;
    int w = 8;
    int q = 6;

    subtrair(x,y);

    System.out.println("Valor de X:" + x);       // X => 5
    System.out.println("Valor de Y:" + y);       // Y => 10

    subtrair(w,q);
    System.out.println("Valor de W:" + w);       // W => 8
    System.out.println("Valor de Q:" + q);       // Q => 6
  }

  public static void subtrair(int a , int b) {
    a = a - 2;
    b = b - 1;
  }
}
```

Neste exemplo o método *subtrair* possui 2 parâmetros: a e b. Quando o método é executado no programa principal (método main), deverão ser utilizados dois valores como parâmetros na chamada do método. O primeiro valor passado será transferido para o primeiro parâmetro do método (parâmetro a) , e o segundo valor passado será transferido para o segundo parâmetro do método (parâmetro b).

Na primeira execução (subtrair(x,y);) o valor da variável x é o primeiro valor passado, portanto, este valor será transferido para o parâmetro a, e o valor da variável y é o segundo valor passado, portanto, será transferido para o parâmetro b.

Na segunda execução (subtrair(w,q);) o valor da variável w é o primeiro valor passado, portanto, este valor será transferido para o parâmetro a, e o valor da variável q é o segundo valor passado, portanto, será transferido para o parâmetro b.

Entretanto, após a execução do método subtrair, os valores das variáveis passados como parâmetro (x e y , w e q) não sofreram nenhuma alteração, porque no método a subtração é feita utilizando os parâmetros a e b, que são variáveis locais, e isto não afeta os valores originais de x,y,w e q.

Para que um método altere o valor de uma variável global, utilizada na passagem de parâmetro, o parâmetro tem que ser declarado como ***parâmetro por referência***.

9.5 – Parâmetros por Valor e por Referência

Quando em um método um parâmetro e passado por Valor, o valor contido na variável utilizada na passagem de parâmetro pelo programa que executou o método é copiado para a variável de parâmetro do método. Desta forma, as duas variáveis, apesar de terem o mesmo valor, são independentes e representam valores alocados em locais diferentes da memória.

No caso e um parâmetro ser passado por Referência, a variável de parâmetro do método recebe apenas um apontamento para o valor armazenado na memória pela variável utilizada na passagem de parâmetro pelo programa que executou o método. Ou seja, apesar de serem duas variáveis, elas apontam para um mesmo valor alocado na memória. Desta forma, se o valor de uma das variáveis for alterado, a alteração ocorrerá na verdade, nas duas.

No Java, toda vez que uma variável de tipo primitivo (como: *int, float, doulble, long, char, boolean, byte*) for utilizada como parâmetro, será um parâmetro por Valor. E os tipos não primitivos (como: vetor, matriz, classes e outros objetos) são sempre parâmetros por referência.

Exemplo 3:

```
import java.util.*;
public class Exemplo3 {

  public static void main(String Args[]) {

    int x[] = newint[1];
    int y[] = newint[1];
    int w[] = newint[1];
    int q[] = newint[1];

    x[0] = 5;
    y[0] = 10;
    w[0] = 8;
```

```
      q[0] = 6;

      subtrair(x,y);
      System.out.println("Valor de X:" + x[0]);    // X => 3
      System.out.println("Valor de Y:" + y[0]);    // Y => 9
      subtrair(w,q);
      System.out.println("Valor de W:" + w[0]);    // W => 6
      System.out.println("Valor de Q:" + q[0]);    // Q => 5
    }

    public static void subtrair (int a[] , int b[]) {
      a[0] = a[0] - 2;
      b[0] = b[0] - 1;
    }
  }
```

Neste terceiro exemplo, os parâmetros do método (`a` e `b`) são dois vetores, tipos não primitivos do Java que quando utilizados como parâmetros em métodos, serão parâmetros por referência. Em seguida, tanto na primeira chamada do método `subtrair(x,y)` quanto na segunda `subtrair(w,q)` os vetores `x,y,w` eqsão passados como parâmetros sem os colchetes contendo uma posição específica do vetor, isto indica que o vetor inteiro está sendo passado como parâmetro para o método. Desta forma, após a execução do método `subtrair(x,y)` e `subtrair(w,q)`, os valores dos vetores passados como parâmetro (`x` e `y` , `w` e `q`) sofrerão as mesmas alterações que os respectivos vetores parâmetros `a` e `b`sofrerem dentro do método. Sendo assim, a subtração feita no primeiro parâmetro (`a`) será também feita em `x`e `w`. E a subtração feita no segundo parâmetro (`b`) será feita da mesma forma em `y` e `q`.

9.6 – Métodos com retorno de valor

Os métodos que retornam valores para o programa que os chamou possuem antes do fim o comando de retorno de valor *return <valor>*. O tipo do valor que o método retornará deve ser especificado na declaração do método no mesmo local onde entrava a palavra *void* para os métodos que não retornam valor.

Exemplo 4:

```
public class Exemplo4{
  public static void main(String Args[]){
    int valor;
    Scanner leia = new Scanner(System.in);
    System.out.print("Digite um valor inteiro para o fatorial:");
    valor = leia.nextInt();
    System.out.print("Fatorial de " + valor + " é:" + calcularFatorial(valor));
  }
  public static long calcularFatorial(int n){
    long fat;
    int i;
    fat = 1;
    for ( i = 1; i <= n ; i++ ) {
      fat = fat * i;
    }
    return fat;
  }
}
```

Neste exemplo, o método `calcularFatorial`, possui o tipo `int`, isto significa que retornará um valor `int`(inteiro) ao programa que o executou.

Neste exemplo a execução do método está contida dentro do próprio comando `System.out.println`, onde o valor exibido na tela será o resultado retornado pelo método.

Repare o comando `return` na ultima linha do método representando o momento que o método está retornando o valor calculado para o programa que o executou.

9.7 – Métodos gravados em arquivos externos

No Java é possível utilizar métodos que foram escritos em classes diferentes daquela onde será necessária a sua utilização, possibilitando assim a reutilização de métodos entre classes (programas) diferentes. Pode-se também criar um arquivo que funcionaria como uma biblioteca de sub-rotinas com uma classe de nome `Rotinas` contendo vários métodos que poderão ser utilizados em qualquer outra classe.

No exemplo 5 abaixo, a primeira parte de código se refere ao programa Java de nome Rotinas.java que contém a classe `Rotinas` e dentro dela o método `calcularFatorial`.

A segunda parte de código se refere ao programa Exemplo5.java que executa o método `calcularFatorial` contido na classe `Rotinas` no programa Rotinas.java.

Exemplo 5 - primeira parte - Rotinas.java

```
import java.util.*;
public class Rotinas {

  public static int calcularFatorial( int n) {
    int i,fat;
    fat = 1;
    for ( i = 1; i <= n ; i++ ) {
      fat = fat * i;
    }
    return fat;
  }
}
```

Exemplo 5 - segunda parte - Exemplo5.java

```
import java.util.*;
public class Exemplo5 {
  public static void main(String Args[]) {
    int valor;
    Scanner leia = new Scanner(System.in);
    System.out.print("Digite um valor inteiro para o fatorial:");
    valor = leia.nextInt();
    System.out.println("O fatorial é:"  +  Rotinas.calcularFatorial(valor));
  }
}
```

EXERCÍCIOS (parte 8):

Exercicio 8.1 - Fazer um programa para receber 3 números inteiros: *numA, numB, numC*. Em seguida, o programa deverá criar os seguintes métodos abaixo:

a) - Fazer um método que receba como parâmetro os números *numA* e *numB*. O método deverá calcular e retornar a soma dos números inteiros existentes entre *numA* e *numB*.

b) - Fazer um método que receba como parâmetro os números *numA*, *numB* e *numC*. O método deverá imprimir os números existentes entre *numA* e *numB* que sejam divisíveis por *numC*.

c) - Fazer um método que receba como parâmetro os números *numA* e *numC*. Considerando que *numA* seja 100%, o método deverá calcular e retornar o valor percentual de *numC* em relação a *numA*.

Exercício 8.2 - Fazer um programa para receber via teclado o Nome do Empregado, o Código do Empregado (numérico), e o Número de Peças fabricadas em um determinado mês. O programa deverá imprimir uma lista, contendo o Nome do Empregado, o Salário e a Média de Salários da empresa. Para calcular o Salário, considerar:

- 1 a 200 peças fabricadas – salário de R$ 2,00 por peça
- 201 a 400 peças fabricadas – salário de R$ 2,30 por peça
- acima de 400 peças fabricadas – salário de R$ 2,50 por peça

No final do relatório, imprimir o total gasto em salários pela empresa.

Consistências:
- O número de peças deverá ser maior que zero.
- O código deverá ser um número entre 1 e 999

Métodos:
- Fazer um método para calcular o Salário do Empregado
    - Parâmetro: número de peças
    - Retorno: salário

Observações:
- Adotar um flag para encerrar a entrada de dados.
- Considerar o máximo de 100 empregados.

Layout do relatório:

```
Nome                             Salário
-------------------------------  ----------
XXXXXXXXXXXXXXXX                  999.99
XXXXXXXXXXXXXXXX                  999.99
XXXXXXXXXXXXXXXX                  999.99
XXXXXXXXXXXXXXXX                  999.99

Total pago com salários: 9,999.99
Média dos salários: 999,99
```

Exercício 8.3 - Fazer um programa para controlar as contas de uma rede de Hoteis. Receber via teclado o Nome do Hóspede, o Dia de Entrada no Hotel, o Dia de Saída do Hotel, o Tipo de Quarto e a Cidade do Hotel.

O programa deverá por meio de um método de nome *calcularConta*, calcular e imprimir o valor da Conta de cada hóspede de acordo com a seguinte tabela:

Tipo de Quarto	Valor da Diária
STANDARD	120,00
LUXO	150,00
SUPER-LUXO	180,00

O valor da conta será o valor da diária multiplicada pelo número de diárias da hospedagem. Para descobrir o número de diárias, subtrair o dia de saída do dia de entrada no hotel.

Fórmula => Valor da Conta = (diaSaida - diaEntrada) * Valor Diaria

Ex:

Dia de Entrada: 20
Dia de Saída: 25
Tipo de Quarto: Luxo - Valor Diaria: 150,00
Valor da conta: (25 – 20) * 150,00 = 750,00

O método deverá receber como parâmetro a Dia da Entrada, o Dia da Saída e o Tipo de Quarto.

O programa deverá imprimir no final um relatório com o Nome do hóspede e o Valor da Conta, de todas as contas acima da média.

Consistências:

- Fazer um método de nome *cidadeEhValida* para consistir a Cidade do Hoteldigitada.
  - Parâmetro: Cidade do Hotel digitada.
  - Este método deverá pesquisar no vetor *vetCidades* se o nome da cidade informada existe lá, caso positivo, o método deverá retornar o valor TRUE, caso negativo retornar o valor FALSE.

- Consistir o tipo de quarto para aceitar somente os valores STANDARD, LUXO ou SUPER-LUXO.
- O Dia de Saída deverá ser maior que o Dia de Entrada.

Obs:

- declarar no programa um vetor global de nome *vetCidades* contendo os seguintes nomes de cidades:
  BELO HORIZONTE, SÃO PAULO, RIO DE JANEIRO, SALVADOR e CURITIBA
- Criar um flag para encerrar a entrada de dados.
- O número máximo de hospedagens que o programa receberá será 100.
- Considerar que a entrada e saída no hotel ocorrem sempre no mesmo mês.

Layout do relatório:

```
Relatório de contas acima da média
Nome do hóspede            Vlr Conta
-----------------------------  ------------
XXXXXXXXXXXXXXXX     999.99
XXXXXXXXXXXXXXXX     999.99
XXXXXXXXXXXXXXXX     999.99
XXXXXXXXXXXXXXXX     999.99
```

10 – Métodos da linguagem Java - Parte 2

10.1 - Parte inteira de um número (typecasting)

```
(int)NUMERO_REAL
```

Retorna a parte inteira do NUMERO_REAL

Exemplo:

```
int valor;
valor = (int) 2.3;        // o valor será 2
valor = (int) 3.85;       // o valor será 3
valor = (int) 243 / 10;   // o valor será 24
```

10.2 – Métodos para manipulação de cadeias de caracteres (Strings e char).

Os métodos para manipulação de strings no Java estão contidos no pacote de classes *java.lang*. Este pacote de classes está contido na base de compilação do Java, sendo assim não necessita de importação por meio do comando *import*.

10.2.1–RETORNAR PARTE DE UMA STRING

```
STRING.substring(posição_inicial,posição_final)
```

O método `substring`retorna a cópia dos caracteres de uma variável `STRING` a partir da `posição_inicial` até a `posição_final - 1`. Caso a `posição_final` não seja informada, será copiado o conjunto de caracteres da `posição_inicial` até o final da `STRING`.

OBS: primeiro caracter de uma String é o de número ZERO.

Exemplo 1:

```
String texto, subTexto1, subTexto2, subTexto3;
texto = "AERODINAMICA";
subTexto1 = texto.substring( 4 ) ;        // subTexto1 = "DINAMICA"
subTexto2 = texto.substring( 0 , 4 );     // subTexto2 = "AERO"
subTexto3 = texto.substring( 2 , 6 );     // subTexto3 = "RODI"
```

OBS: Para facilitar a especificação da posição final, basta somar a posição inicial com o total de caracteres desejados e o resultado será a posição final:

- no caso do segundo comando do exemplo acima, a variável `subTexto2` receberá a substring `"AREO"`. Se a posição inicial é 0 (zero), e o total são 4 caracteres, basta somar 0 + 4, e o resultado (4) é a posição final.
- no caso do terceiro comando, a variável `subTexto3` receberá a substring `"RODI"`, se a posição inicial é 2 e o total são 4 caracteres, basta somar 2 + 4, e o resultado (6) é a posição final.

Exemplo 2 (concatenando dois caracteres de uma String formando outra String):

```
String texto= "AERODINAMICA";
String subTexto;
subTexto = texto.substring(0,1) + texto.substring(4,5);
System.out.print(subTexto); // será exibido na tela o valor AD
```

10.2.2 – RETORNAR UM CARACTER DA STRING

```
STRING.charAt(posição)
```

O método `CharAt` retorna o caracter (tipo `char`) contido na `posição` informada da `STRING`.

Exemplo:

```
String texto = "DINAMICA";
System.out.print( texto.charAt(2) );//será exibido na tela o caracter N
System.out.print( texto.charAt(4) );//será exibido na tela o caracter M
```

10.2.3 – RETORNAR O TAMANHO DE UMA STRING

```
STRING.length()
```

O método `length` retorna o número de caracteres de uma string, ou seja, o tamanho da string em número de caracteres.

Exemplo:

```
String texto1 = "DINAMICA";
String texto2 = "JOSE DA SILVA";
System.out.println( texto1.length() ); // o valor exibido na tela será 8
System.out.println( texto2.length() );  // o valor exibido na tela será 13
```

10.2.4 – CONVERTER NÚMERO EM STRING:

```
String.valueOf(valor_numérico)
```

O método `String.valueOf` retorna uma cópia do `valor_numérico` convertido para o tipo `String`.

Exemplo 1:

```
int numero = 15;
String texto;
texto = String.valueOf(numero);
System.out.println(texto);    // o valor exibido na tela será 15
```

OBS: No exemplo 1, após o uso do método `String.valueOf`, a variável `num` continuará contendo o valor `15`, e a variável `texto` passará a conter o valor `"15"`.

Exemplo 2:

```
String data;
int dia = 10;
int mes = 12;
int ano = 1980;
data = String.valueOf(dia) + '/' + String.valueOf(mes) + '/' +
     String.valueOf(ano);
System.out.print(data);  // o valor exibido na tela será 10/12/1980
```

OBS: No exemplo 2, após o uso do método `String.valueOf` nas variáveis `dia`, `mes` e `ano`, o valor Caracter de cada uma delas será concatenado com as barras `('/')` e o resultado atribuído a variável `data` ficará com o valor `"10/12/1980"`.

10.2.5 – CONVERTER NÚMERO EM CHAR:

`Character.forDigit(``valor_numérico , base_numérica`**`)`**

O método `Character.forDigit` retorna uma cópia do `valor_numérico` convertido para o tipo `Char`.

Exemplo 1:

```
char caracter;
int digito = 5;
caracter = Character.forDigit(digito,10);
System.out.print(caracter);   // o valor exibido na tela será 5
```

OBS: No exemplo 1, após o uso do método `Character.forDigit`, a variável `digito` continuará contendo o valor `5`, e a variável `caracter` passará a conter o valor `'5'`.

10.2.6 – CONVERTER STRING EM NÚMERO:

`Integer.parseInt(``string`**`)`**

O método `Integer.parseInt` retorna uma cópia da `string` convertida para o tipo `int`.

`Float.parseFloat(``string`**`)`**

O método `Float.parseFloat` retorna uma cópia da `string` convertida para o tipo `float`.

`Double.parseDouble(``string`**`)`**

O método `Double.parseDouble` retorna uma cópia da `string` convertida para `double`.

`Byte.parseByte(``string`**`)`**

O método `Byte.parseByte` retorna uma cópia da `string` convertida para o tipo `byte`.

`Long.parseLong(``string`**`)`**

O método `Long.parseLong` retorna uma cópia da `string` convertida para o tipo `long`.

Exemplo 1:

```
String texto= "15";
byte numero;
numero = Byte.parseByte(texto);
System.out.print(numero);        // valor exibido será o número 15
```

Exemplo 2:

```
int numero;
int dia = 10;
String texto = "15";
numero = dia + Integer.parseInt(texto);
System.out.print(numero);    // valor exibido será o número 25
```

Exemplo 3:

```
String texto= "123.34";
double numero;
numero = Double.parseDouble(texto);
System.out.print(numero);    // valor exibido será o número 123.34
```

OBS: Caso a `STRING` contenha algum caracter que impeça a sua conversão para número (como uma letra ou um símbolo - * ? > ...), a utilização do método `parse` provocará um erro e o programa será cancelado por erro em formato numérico (`NumberFormatException`). Para evitar que isto aconteça, todas as vezes que o método `parse` for utilizado sem a certeza que todos os caracteres da `STRING` possam ser convertidos para número, deve-se tratar o erro por meio do comando `try-catch`, como nos exemplos a seguir:

Exemplo 4:

```
String texto;
double numero;
Scanner leia = new Scanner(System.in);
System.out.print("Digite o texto com caracteres númericos: ");
texto = leia.nextLine();

try{
  numero = Double.parseDouble(texto);
  System.out.print(numero);    // será exibido o número digitado
                               // na entrada de dados
}catch (NumberFormatException E) {
   System.out.print("O texto digitado não pode ser convertido em
          número !");
}
```

OBS: No Exemplo 4, o comando `try` testa a execução do método `Double.parseDouble`:

- se o método puder ser executado (ou seja, se a variável `texto` puder ser convertida em número), serão executados todos os comandos que estão dentro do `try`, ou seja, a variável `numero` receberá o `texto` convertido para um número real e o comando `System.out.print(numero)` será executado em seguida.

- se o método não puder ser executado (ou seja, se a variável `texto` não puder ser convertida e número), somente os comandos que estão dentro catch serão executados, ou seja, somente o

comando `System.out.print("O texto digitado não pode ser convertido em número !")` será executado.

Exemplo 5 (consistência):

```
String texto;
int numero;
boolean valido;
Scanner leia = new Scanner(System.in);
do{
  System.out.print("Digite o texto com caracteres númericos: ");
  texto = leia.nextLine();

  try{
    numero = Integer.parseInt(texto);
    System.out.print(texto);       // exibir o número digitado
    valido = true;
  }catch (NumberFormatException E){
    System.out.println("O texto digitado não pode ser convertido em
          número ! Digite Novamente");
    valido = false;
  }
}while ( ! valido);
```

10.2.7 – CONVERTER CHAR EM NÚMERO:

`Character.digit(``char , base_numérica`**`)`**

O método `character.Digit` retorna uma cópia do caracter `char` convertido para o tipo `int` na `base_numérica` informada. Neste caso, os únicos resultados possíveis são os números com apenas um dígito: de zero a nove (0...9). Isto porque uma expressão caracter só pode ter um único caracter, então não poderia resultar em um número com mais de um dígito. Nem mesmo um sinal negativo poderia existir, neste caso, os números retornados (0...9) serão sempre positivos.

Exemplos de bases numéricas:
- 10 para base decimal;
- 2 para base binária;
- 16 para base hexadecimal;

Para o método `Character.digit`, se o caracter `char`não for um dos dígitos de 0 a 9, o resultado da conversão será -1, indicando que a conversão não pode ser feita.

Exemplo 1:

```
int numero;
char caracter = '3';
numero = Character.digit(caracter,10);     // conversão na base decimal
System.out.println(numero);              // o valor exibido será o número 3
```

Exemplo 2:

```
int numero;
char caracter = 'R';
numero = Character.digit(caracter,10);      // conversão na base decimal
System.out.println(numero);                 // o valor exibido será o número -1
```

OBS: No Exemplo 2, o caracter `'R'` não pode ser convertido para um número decimal, sendo assim, a variável `numero` recebeu como retorno da conversão o valor `-1`, indicando que a conversão não é possível.

10.3 - Outros métodos de manipulação de Strings:

10.3.1 – COMPARAR SE UMA STRING É MAIOR QUE A OUTRA

```
STRING_1.compareTo(STRING_2)
```

O método *compareTo* compara a grandeza caracter entre strings considerando a posição de cada caracter das strings na tabela ASCII.

O método retornará um valor positivo se `STRING_1` for maior que `STRING_2` em relação a posição dos caracteres de cada string na tabela ASCII, ou seja, os caracteres da `STRING_1` vem depois dos caracteres da `STRING_2` na Tabela ASCII.

O método retornará um valor negativo se `STRING_1`for menor que `STRING_2`, e retorna 0 (zero) se as duas strings forem iguais. exemplo:

OBS: na tabela ASCII, as letras maiúsculas vêm antes das minúsculas, sendo assim, as letras minúsculas são consideradas maiores que as minúsculas. Ex: "a" e maior que "A".

Exemplo 1:

```
String textoA = "21" ;
String textoB = "05";
int result = textoA.compareTo(textoB);
System.out.println(result);//como o primeiro caracter de textoA vem depois do primeiro caracter do textoB,
                           //  textoA é maior que textoB, sendo assim o valor de result será maior que zero.
```

Exemplo 2:

```
String textoA = "JORGE" ;
String textoB = "JOSE";
int result = textoA.compareTo(textoB);
System.out.println(result);//como o 1o e 2o caracteres das duas strings são iguais, a diferença ficará a partir
                           //do 3o. caracter.  Como o 3o caracter de textoA vem antes do 3o. caracter do
                           //textoB, textoA é menor que textoB,sendo assim o valor de result será negativo
```

Exemplo 3:

```
String textoA = "hoje" ;
String textoB = "Hoje";
int result = textoA.compareTo(textoB);
System.out.println(result);//as strings são iguais exceto pelo primeiro caracter.Como o primeiro caracter de
                           //  textoA é minúsculo, vem depois do primeiro caracter do textoB, ou seja,
                           //  textoA é maior que textoB, sendo assim o valor de result será maior que zero.
```

10.3.2 – COMPARAR SE UMA STRING É MAIOR QUE A OUTRA (sem maiúsculo / minúsculo)

```
STRING_1.compareToIgnoreCase(STRING_2)
```

O método *compareToIgnoreCase* funciona exatamente da mesma forma que o *compareTo*. Entretanto, quando existir caracteres que são letras nas strings, a letra maiúscula será considerada igual a letra minúscula.

Exemplo 1:

```
String textoA = "hoje" ;
String textoB = "Hoje";
int result = textoA.compareToIgnoreCase(textoB);
System.out.println(result);//  neste caso as as strings são iguais já que maiúsculo/minúsculo será ignorado,
                           //  sendo assim o valor de result será zero.
```

11 – Entrada e Saída de dados utilizando a classe JOptionPane do SWING (GUI – Graphical User Interface) ao invés da classe Scanner.

Swing é uma biblioteca gráfica oficial inclusa no Java em qualquer JRE ou JDK para a construção de interfaces gráficas. Para utilizar o Swing no Java basta importar o pacote Swing que contém as Classes e Métodos necessários:

```
import javax.swing.*;
```

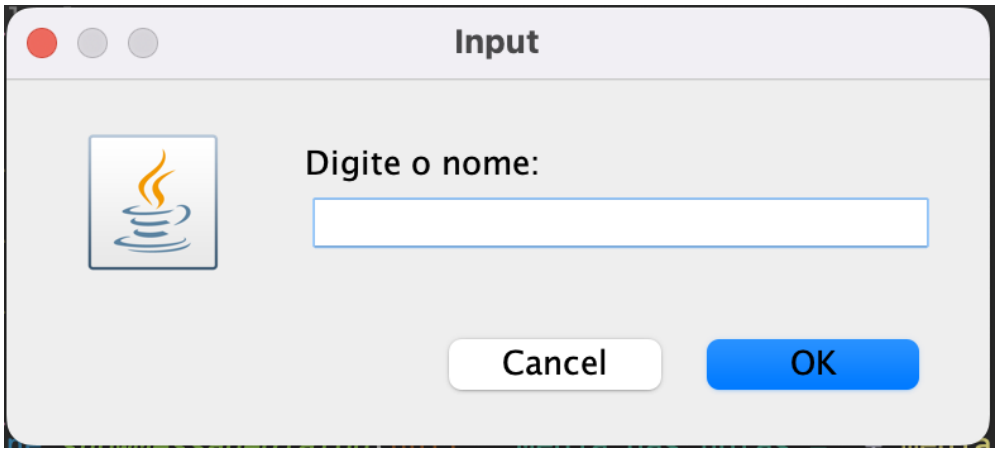
Neste exemplo usaremos a classe JOptionPane para fazer a Entrada e Saída de dados:

11.1 - Para a Entrada de dados podemos usar o método `showInputDialog`:

`Exemplo 1:`

```
import javax.swing.*;
public class Exemplo_JOptionPane {
      public static void main(String[] args) {

            String nomeAluno;
            nomeAluno = JOptionPane.showInputDialog("Digite o nome: ");
        }
}
```

Após esta linha de comando, será apresentada uma Janela com a mensagem e uma caixa de texto para o usuário digitar o nome do Aluno:



Exemplo 2:

Para receber valores digitados que não sejam textos, números por exemplo, teremos que utilizar o método ***parse*** que permite converter valores do tipo String para números.

```
import javax.swing.*;
public class Exemplo_JOptionPane {
      public static void main(String[] args) {

            String nomeAluno;
            int nota;
```

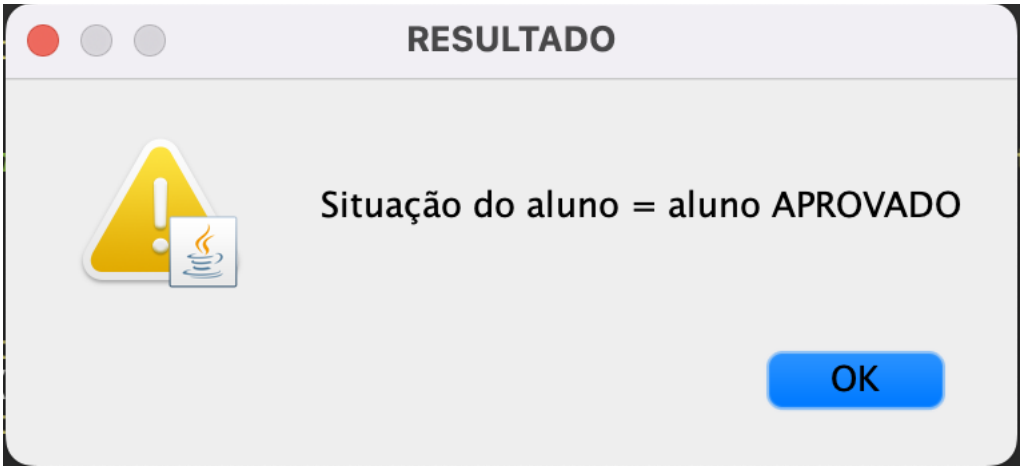
```
            nomeAluno = JOptionPane.showInputDialog("Digite o nome: ");
            nota = Integer.parseInt(JOptionPane.showInputDialog ("Digite a
nota"));
      }
}
```

Neste exemplo acima, o método **Integer.*parseInt*** vai converter o valor ***String*** digitado pelo usuário por meio do **JOptionPane.*showInputDialog*** em um valor do tipo ***int***.

11.2 - Para a Saída de dados pode ser usado o método *showMessageDialog*:

```
      if (nota <= 4){
            resultado = "aluno REPROVADO";
      }else if (nota1 < 6){
            resultado = "aluno EM RECUPERACAO";
      }else{
            resultado = "aluno APROVADO";
      }

      JOptionPane.showMessageDialog(null, "Situação do aluno = " +
resultado, "RESULTADO", JOptionPane.WARNING_MESSAGE);
```

Neste exemplo acima, o método *showMessageDialog* irá apresentar em uma janela o texto apresentado como segundo parâmetro do método e com o título apresentado como terceiro parâmetro no método.



O código completo deste exemplo seria:

```
import javax.swing.*;

public class Exemplo_JOptionPane {

      public static void main(String[] args) {
            // 1 - variaveis
            String nomeAluno;
            int nota;
```

```
            String resultado;

            // 2 - Entrada de dados

            nomeAluno = JOptionPane.showInputDialog("Digite o nome: ");
            nota = Integer.parseInt(JOptionPane.showInputDialog ("Digite a
primeira nota"));

            // 3 - calculos

            if (nota <= 4){
                  resultado = "aluno REPROVADO";
            }else if (nota < 6){
                  resultado = "aluno EM RECUPERACAO";
            }else{
                  resultado = "aluno APROVADO";
            }

      JOptionPane.showMessageDialog(null, "Situação do aluno = " +
resultado, "RESULTADO", JOptionPane.WARNING_MESSAGE);

      }
}
```

12 – Criando janelas com o SWING.

12.1 – Introdução

Java Swing é uma biblioteca gráfica para criação de interfaces gráficas de usuário (GUI) em aplicações Java. Ela fornece um conjunto de componentes de interface de usuário que podem ser usados para criar janelas, botões, caixas de diálogo, barras de menu e muito mais. Swing é parte da plataforma Java Standard Edition (Java SE) e é uma das principais escolhas para desenvolvimento de GUI em Java.

Neste documento, você será introduzido aos conceitos básicos do Java Swing, aprenderá como criar uma aplicação Swing simples e verá exemplos de código para criar componentes de interface comuns.

12.2 - Componentes básicos do Swing.

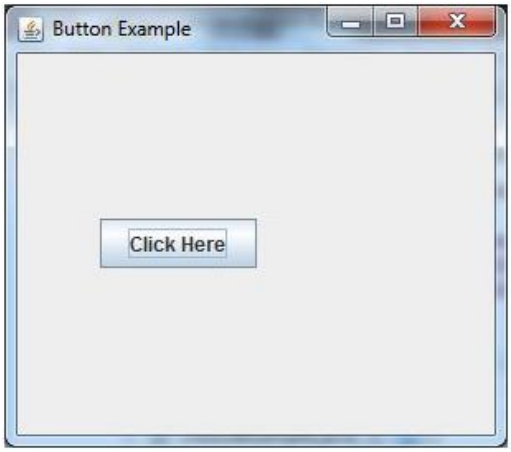
Os componentes Swing são objetos que podem ser adicionados a um contêiner (como uma janela) para criar a interface do usuário. Alguns dos componentes Swing mais comuns incluem:

1. **JFrame**: Classe que permite criar uma janela de nível superior que pode conter outros componentes Swing. É usada como a janela principal da aplicação.

O exemplo acima e a direite é um *JFrame* configurado com o layout *BorderLayout* que define um contêiner, organizando e redimensionando seus componentes para caber em cinco regiões: norte, sul, leste, oeste e centro.

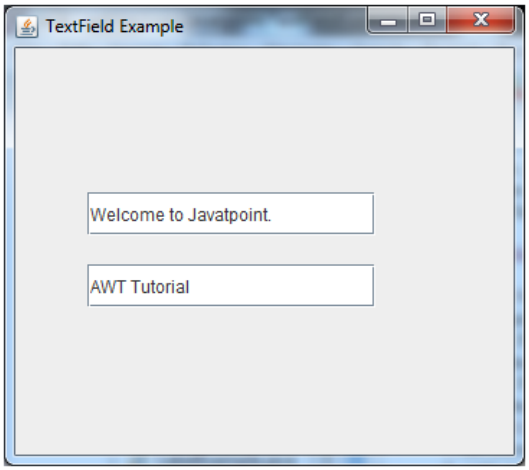
2. JPanel: Classe que permite criar um contêiner que pode ser adicionado a um JFrame. É útil para agrupar outros componentes.

3. **JButton**: Classe que permite criar um botão clicável que pode ser usado para acionar ações quando pressionado.



4. **JLabel**: Classe que permite exibir um texto ou uma imagem estática em um JFrame.

5. **JTextField**: Classe que permite criar uma caixa de texto para receber a digitação do usuário.

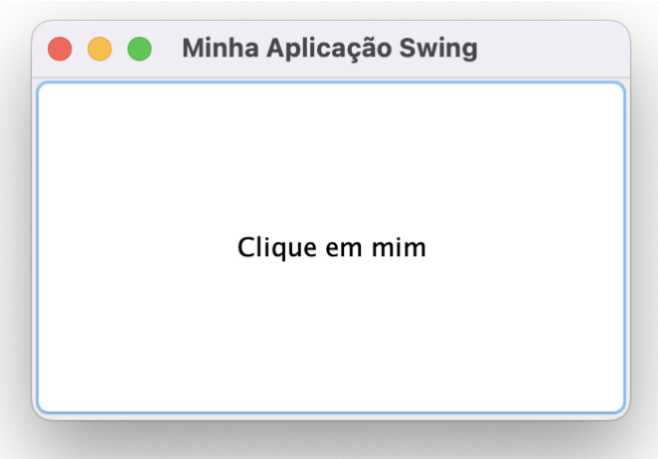


12.3 - Exemplo Básico

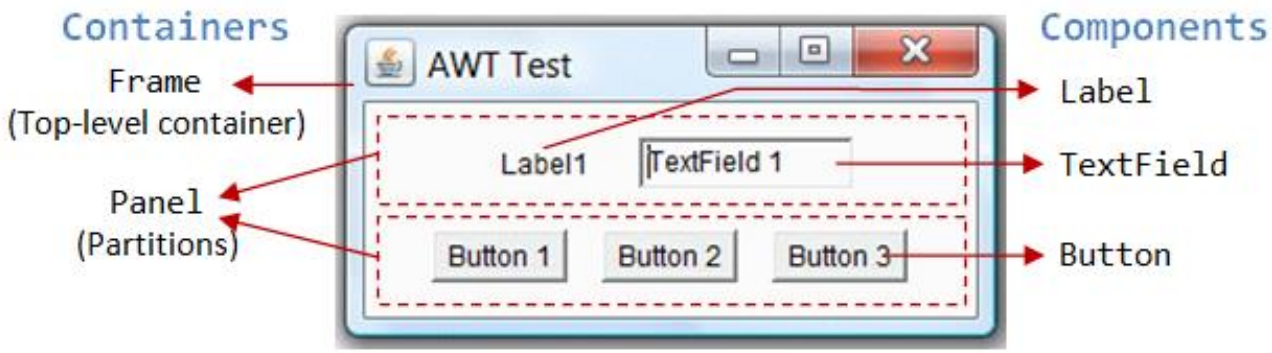
Aqui está um exemplo básico de como criar uma janela JFrame simples com um botão JButton:

```
import javax.swing.*;

public class MinhaAplicacaoSwing {

    public static void main(String[] args) {

        // Cria uma instância de JFrame
        JFrame jframe = new JFrame("Minha Aplicação Swing");

        // Cria um botão
        JButton botao = new JButton("Clique em mim");

        // Adiciona o botão ao JFrame
        jframe.add(botao);

        // Define o tamanho da janela
        jframe.setSize(300, 200);

        // Define o comportamento padrão de fechamento
        jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Torna a janela visível
        jframe.setVisible(true);
    }
}
```

Abaixo a janela criada pelo código acima:



O Java Swing é uma biblioteca poderosa para a criação de interfaces gráficas em Java. Este documento introduziu você aos conceitos básicos do Swing, mostrou alguns dos componentes mais comuns e forneceu um exemplo simples de criação de uma aplicação Swing. À medida que você se aprofunda no desenvolvimento de GUI em Java, você aprenderá a usar mais componentes e recursos avançados do Swing para criar interfaces de usuário sofisticadas e interativas.


Containers
Frame
(Top-level container)
Panel
(Partitions)
AWT Test
Label1
TextField 1
Button 1
Button 2
Button 3
Components
Label
TextField
Button

EXERCÍCIOS (parte 9):

Exercício 9.1 – Faça um programa em Java para receber via teclado a data de nascimento de uma pessoa e a data de hoje, ambas no formato string DD/MM/AAAA. O programa deverá criar um método para calcular e imprimir a idade da pessoa.

Exercício 9.2 – Fazer um programa em Java para receber via teclado um código caracter contendo 11 dígitos. Os 9 primeiros dígitos representam o código em si e os 2 últimos os dígitos representam os dígitos verificadores. Utilizando a regra de cálculo dos dígitos verificadores, o programa deverá conter um método para calcular os dois dígitos verificadores do código e comparar com os 2 dígitos verificadores digitados no código. Se forem iguais, o programa deverá imprimir a mensagem: "Dígito Correto", caso contrário imprimir "Dígito Inválido".

O cálculo dos dois dígitos verificadores deverá ser feito baseando-se nos 9 primeiros dígitos do Código:

1º. Dígito verificador:
- 1 - somar entre si o valor de cada dígito do código
- 2 - dividir o resultado por 10
- 3 - o 1º. Dígito será a parte inteira do resultado da divisão do item 2 (acima).

2º. Dígito verificador:
- 1 - multiplicar entre si o valor de cada dígito do código
- 2 - o 2º. Digito verificador será o ultimo algarismo a direita do resultado da multiplicação do item 1 (acima).

Exemplo:

9 primeiros dígitos do Código: "821324312"

1º. Digito verificador: 8+2+1+3+2+4+3+1+2 = 26 / 10 = 2,6 => Parte inteira = **2**

2º. Digito verificador: 8*2*1*3*2*4*3*1*2 = 2304 => Ultimo algarismo a direita = **4**

OBS: - consistir a entrada de dados para aceitar somente CODIGO:
- com 11 caracteres
- todos os 11 caracteres devem ser dígitos

Exercício 9.3 - Fazer um programa em Java para calcular uma conta telefônica. Serão digitadas via teclado o HORÁRIO INICIAL e o HORÁRIO FINAL de cada ligação. Calcular e imprimir o CUSTO de cada chamada de acordo com a tabela abaixo:

Intervalo de horário	Custo do minuto
00:00 ás 05:59	R$ 0,10
06:00 às 07:59	R$ 0,15
08:00 às 17:59	R$ 0,20
18:00 às 23:59	R$ 0,15

Obs:

- Receber os horários em variável do tipo String no formato HH:MM. Exemplo: "10:30";

- Imprimir o valor TOTAL DA CONTA TELEFÔNICA;

- Considerar que as ligações sempre ocorrem dentro do mesmo dia;

- Considerar o valor do minuto relativo aHORÁRIO INICIAL da chamada para calcular o CUSTO da chamada. Ex: se a chamada começou as 06:30, o valor do minuto para toda a chamada será R$0,15 independente de quanto tempo durar a chamada;

- Criar um método de nome *horaEhValida* que deverá receber como parâmetro uma hora no formato HH:MM e consistir se HH está entre 0 e 23 e MM está entre 0 e 59. Caso afirmativo, o método deverá retornar o valor TRUE, caso negativo o método deverá retornar o valor FALSE.

- Criar uma consistência para que o HORÁRIO FINAL da chamada seja sempre maior que o HORÁRIO INICIAL.

- Adote um Flag para encerrar a entrada de dados.

Exercício 9.4 - O DETRAN deseja fazer o controle das multas de veículos. Faça um programa em Java que receba via teclado a PLACA DO VEÍCULO, a DATA DA MULTA (DD/MM/AAAA) e o VALOR DA MULTA.

O programa deverá consistir a entrada de dados da seguinte forma:

- A placa deverá ser uma String de 7 caracteres e ser formada por três letras e quatro dígitos. Ex: GVP5566

- O valor da multa deverá ser maior que zero.

- Fazer um método de nome *dataEhValida* para consistir a data da multa:
    - o método deverá receber como parâmetro uma data no formato DD/MM/AAAA
    - a consistência deverá seguir as seguintes regras:
        - a String deverá ter 10 caracteres de tamanho.
        - o 3º. E o 6º. Caracteres deverão ser uma barra ('/').
        - para os meses de Janeiro, Março, Maio, Julho, Agosto, Outubro e Dezembro o dia deverá ser entre 1 e 31.
        - para os meses Abril, Junho, Setembro e Novembro o dia deverá ser entre 1 e 30.
        - para o mês de Fevereiro:
            - anos divisíveis por 4 e não divisível por 100 ou anos divisíveis por 400 o dia deverá ser entre 1 e 29 ((ano bissexto);
            - para os demais anos o dia deverá ser entre 1 e 28;
        - os mês deverá ser entre 1 e 12
        - o ano deverá ser menor ou igual ao ano atual.
    - o método deverá retornar um valor do tipo Boolean. Caso a data recebida como parâmetro esteja de acordo com as regras acima a função retornará o valor TRUE, caso contrário, retornará o valor FALSE.

Como resultado final o programa deverá imprimir:
- A soma dos valores das multas.
- O valor médio das multas.
- O valor da menor multa.

Obs:
- Definir um Flag para encerrar o programa.

13 - Métodos para manipular Strings - Parte 3

13.1 –RETORNAR UM CARACTER DA TABELA ASCII

`(char)NÚMERO`

Fazer um *type casting* para o tipo *char* em um número ou em uma variável numérica retornará o caracter correspondente ao código decimal daquele `NÚMERO` na tabela ASCII. A tabela ASCII contém 256 caracteres onde o primeiro é o de número 0 e o ultimo o de número 255.

Exemplo:

```
int numero;
numero = 100;
System.out.println((char)numero);   //  exibirá na tela o caracter  d  , que é o caracter
                                    //  de número 100 na tabela ASCII
System.out.println((char)42);  //  exibirá na tela o caracter  * , que é o caracter
                               //  de número 42 na tabela ASCII
```

13.2 –RETORNAR O NÚMERO DE UM CARACTER DA TABELA ASCII

`(int)CARACTER`

O método `int` retorna o número do `CARACTER` na tabela ASCII.

Exemplo:

```
char caracter;
caracter = 'd';
System.out.println((int)caracter);  //exibirá na tela o número 100, que é o número
                                    //  do caracter  d  na tabela ASCII
System.out.println((int)'*');       //  exibirá na tela o número 42, que é o número
                                    //  do caracter  *  na tabela ASCII
```

13.3 –TRANSFORMAR AS LETRAS DE UMA STRING EM MAIÚSCULAS

`STRING.toUpperCase()`

O método `toUpperCase` quando aplicado em uma String, retorna a mesma String contendo todos os caracteres que eram letras minúsculas transformados em letras maiúsculas. Aqueles caracteres da `STRING` que não eram letras ou já eram letras maiúsculas, permanecerão sem nenhuma modificação.

Exemplo 1:

```
String nomeMaiusculo, nomeDigitado;
Scanner leia = new Scanner(System.in);
System.out.print("Digite o Nome: ");
nomeDigitado = leia.nextLine();
nomeMaiusculo = nomeDigitado.toUpperCase();
System.out.println("Nome em letras maiúsculas: " + nomeMaiusculo);
```

Exemplo 2:

```
String nome;
Scanner leia = new Scanner(System.in);
System.out.print("Digite o Nome: ");
nome = leia.nextLine().toUpperCase();
// o nome digitado é atribuído para a variável nomejá transformado em letras maiúsculas
System.out.println("Nome em letras maiúsculas: " + nome);
```

13.4 – TRANSFORMAR A LETRA DE UM CHAR EM MAIÚSCULAS

Character.toUpperCase(CARACTER**)**

No método `Character.toUpperCase` se o `caracter` passado como parâmetro for uma letra minúscula, o método retorna a respectiva letra em maiúsculo. Caso o `caracter` não seja letra ou já é uma letra maiúscula, o método retornará o mesmo `caracter` sem nenhuma modificação.

Exemplo 1:

```
char sexo;
Scanner leia = new Scanner(System.in);
System.out.print("Digite o Sexo: ");
sexo = leia.next().charAt(0);

if (Character.toUpperCase(sexo) !='M' && Character.toUpperCase(sexo) !='F'){
  System.out.print("Sexo Inválido ! ");
}else{
  System.out.println("Parabéns, você digitou um sexo válido ! ");
}
```

Exemplo 2:

```
char sexo;
Scanner leia = new Scanner(System.in);
System.out.print("Digite o Sexo: ");
sexo = leia.next().charAt(0);

sexo = Character.toUpperCase(sexo);

if ( sexo != 'M' && sexo != 'F' ) {
  System.out.print("Sexo Inválido ! ");
}else{
  System.out.println("Parabéns, você digitou o sexo " + sexo +
          " que é válido !");
}
```

13.5 – VERIFICAR SE UMA STRING ESTÁ CONTIDA EM OUTRA

```
STRING_1.contains(STRING_2)
```

O método `contains` retorna `true` se o conjunto de caracteres da `STRING_1` contém o conjunto de caractere da `STRING_2`, caso contrário, retorna `false`.

Exemplo:

```
String texto1 = "FUMEC";
String texto2 = "UME";
String texto3 = "UMA";
if (texto1.contains(texto2)) {     // neste caso o este lógico retornará true
  System.out.println("Variável texto1 contem o conteúdo de texto2");
}

if (texto1.contains(texto3)) {    // neste caso o teste lógico retornará  false
  System.out.println("texto1 contem o conteúdo de texto3");
}else{
  System.out.println("texto1 NÃO contem o conteúdo de texto3");
}
```

13.6 – RETORNA A POSIÇÃO EM QUE UMA STRING ESTÁ DENTRO DE OUTRA

```
STRING_1.indexOf(STRING_2)
```

O método `indexOf` retorna o número da posição inicial da `STRING_2` contida dentro da `STRING_1`. O primeiro caracter de uma String é o de número 0 (zero). Se por acaso o conjunto de caracteres da `STRING_2` não estiver contido na `STRING_1`, o valor retornado será -1.

Exemplo:

```
int posicao;
String texto1 = "FUMEC";
String texto2 = "UME";
String texto3 = "UMA";
String texto4 = "M";
posicao = texto1.indexOf(texto2);       //variável posicao receberá o valor 1
System.out.println(posicao);
posicao = texto1.indexOf(texto4);       //variável posicao receberá o valor 2
System.out.println(posicao);
posicao = texto1.indexOf(texto3);       //variável posicao receberá o valor -1
System.out.println(posicao);
```

13.7 – TROCA UM SEQUÊNCIA DE CARACTERES POR OUTRA

```
STRING.replace(CARACTERES_ANTIGOS,CARACTERES_NOVOS)
```

O método `replace` retorna uma cópia da `STRING` substituindo todas as sequencias de `CARACTERES_ANTIGOS` encontrados pela sequencia de `CARACTERES_NOVOS`.

Exemplo 1:

```
String texto = "JOSE DE CASTRO DE MARIA DE JESUS";
texto = texto.replace(" DE" , "");
System.out.println(texto);  // valor exibido: JOSE CASTRO MARIA JESUS
```

Neste Exemplo 1, após o uso do método `replace`, o conteúdo de `texto` passará a ser `"JOSE CASTRO MARIA JESUS"`, ou seja, as sequencias de caracteres contendo `"DE"` encontrados em `texto` foram substituídos por ausência de caracter `("")`, ou seja, foram excluídas;

Exemplo 2:

```
String texto = "25-10-2012";
texto = texto.replace("-" , "/");
System.out.println(texto); // valor exibido será 25/10/2012
```

Neste Exemplo 2, após o uso do método `replace`, o conteúdo de `texto` passará a ser `"25/10/2012"`, ou seja, as sequencias de caracteres contendo `"-"` encontrados em `texto` foram substituídos pelo caracter `"/"`

13.8 – TROCA A PRIMEIRA SEQUÊNCIA DE CARACTERES POR OUTRA

```
STRING.replaceFirst(CARACTERES_ANTIGOS,CARACTERES_NOVOS)
```

O método `replaceFirst` retorna uma cópia da `STRING` substituindo a primeira sequencia de `CARACTERES_ANTIGOS` encontrados pela sequencia de `CARACTERES_NOVOS`.

Exemplo 1:

```
String texto = "JOSE DE CASTRO DE MARIA DE JESUS";
texto = texto.replaceFirst(" DE" , "");
System.out.println(texto);  // exibido: JOSE CASTRO DE MARIA DE JESUS
```

Neste Exemplo 1, após o uso do método `replaceFirst`, o conteúdo de `texto` passará a ser `"JOSE CASTRO DE MARIA DE JESUS"`, ou seja, apenas a primeira sequencia de caracteres `"DE"` encontrado em `texto` foi substituída por ausência de caracter `("")`, as demais sequencias `"DE"` permaneceram como estavam;

Exemplo 2:

```
String texto = "25-10-2012";
texto = texto.replaceFirst("-" , "/");
System.out.println(texto); // valor exibido será 25/10-2012
```

Neste Exemplo 2, após o uso do método `replaceFirst`, o conteúdo de `texto` passará a ser `"25/10-2012"`, ou seja, apenas o primeirocaracter `"-"`encontrado em `texto` foi substituído por `"/"`

13.9 – TRANSFORMA UMA STRING EM UM VETOR DE CHAR

`STRING.`**`toCharArray()`**

O método `toCharArray` retorna todos os caracteres da `STRING` em um vetor do tipo `char`, contendo cada caracter da `STRING` em um posição do vetor.

Exemplo 1:

```
char vetorDeCaracteres[];
String texto = "DINAMICA";
vetorDeCaracteres = texto.toCharArray();
//  o comando acima copia o conteúdo da String TEXTO para o vetor vetorDeCaracteres
//  colocando cada caracter em uma posição do vetor

System.out.print(vetorDeCaractees[6]);  //  Exibirá na tela o 6o. caracter do vetor => C
```

No exemplo 1, `vetorDeCaracteres` é um vetor do tipo char que recebeu o seguinte conteúdo:

Posição no vetor →	0	1	2	3	4	5	6	7
Conteúdo →	'D'	'I'	'N'	'A'	'M'	'I'	'C'	'A'

EXERCÍCIOS (parte 10):

Exercício 10.1 – Fazer um programa para receber via teclado a digitação em letras minúsculas de um NOME do tipo string. Em seguida o programa deverá executar os seguintes métodos:

1.1 – Criar um método que para Converter a primeira letra do Nome para maiúsculo e retornar o nome convertido.

1.2 - Criar um método para Converter a primeira letra de cada palavra do Nome para maiúscula e imprimir a frase convertida.

1.3 – Criar um método para eliminar espaços em branco digitados à esquerda do Nome e retornar o nome sem os espaços.

1.4 – Criar um método para eliminar espaços em branco digitados à direita do Nome e imprimir o nome sem os espaços.

1.5 – Criar um método para eliminar espaços em branco excessivos digitados entre as palavras do nome de tal forma que fique somente um espaço entre cada palavra, e retornar o nome.

Exercício 10.2 – Fazer um programa em Java que receba via teclado a digitação de até 30 nomes de pessoas.

Para cada um dos nomes digitados o programa deverá conter um método para gerare retornar um Login e uma Senha.

Após gerados o login e senha o programa deverá exibi-los na tela.

O Login será a formado da concatenação da primeira letra de cada nome em maiúsculo. E a senha será formada da concatenação do primeiro dígito do valor ASCII Decimal de cada letra do Login.

Ex: nome digitado: jose maria alves dos santos
Login gerado: JMADS(códigos ASCII decimais: J=**7**4, M=**7**7, A=**6**5, D=**6**8, S=**8**3)
Senha gerada: 77668

Obs: - Consistir a entrada de dados para que o nome da pessoa digitado:
- tenha o tamanho mínimo de 15 caracteres;
- não deve existir caracteres espaço antes do primeiro nome;
- deve existir pelo menos 1 nome e 1 sobrenome;
-deve existir apenas 1 espaço entre o nome e sobrenomes;
- só possua letras em cada nome;

- Criar um FLAG para encerrar a entrada de dados.

14 – Armazenamento de dados em Arquivos.

Registro:

- É um Conjunto de dados heterogêneos, ou seja, dados diferentes, porém relacionados a um mesmo objeto.

Exemplo: Registro de Alunos da FUMEC, que contém os seguintes dados:

- Nome do Aluno
- Endereço do Aluno
- Data de Nascimento
- Sexo
- ...

Ou seja, são dados diferentes, mas todos relativos ao mesmo aluno.

Arquivo de Dados:

- Conjunto de registros seqüênciais armazenados em um dispositivo de armazenamento (ex: HD), gerenciado pelo Sistema Operacional, e manipulado através de programas que podem incluir, alterar, consultar ou excluir dados nos registros.

Exemplo de um arquivo de registros:

- Arquivo que contém os registros dos Alunos da FUMEC:

Ativo	Matrícula	Nome Aluno	Data Nascimento	Mensalidade	Sexo
S	054689	JOSE DA SILVA	10/10/1980	750,00	M
S	025478	MARIA JOSÉ	20/12/1989	830,00	F
N	009871	SAULO JORGE	12/02/1992	690,00	M
S	009871	SAULO SANTOS JORGE	22/02/1992	590,00	M
	(end of file)				

Considerações para manipular registros em arquivos de dados binários:

➔ As inclusões de novos registros são feitas sempre após o ultimo registro do arquivo, em uma área chamada de END OF FILE (final de arquivo)

➔ A exclusão de registros fisicamente no arquivo não é uma operação simples devido ao controle binários dos dados armazenados. Desta forma, uma opção para a exclusão física seria criar um novo arquivo (com nome diferente) e fazer um programa para copiar apenas os registros desejados para este novo arquivo, não copiando os registros não desejados. Após isto, excluir pelo sistema operacional o arquivo anterior e renomear o novo arquivo para o nome original daquele que foi excluído.

➔ A forma mais utilizada, mais segura e eficiente para excluir registros é a exclusão lógica. Para excluir logicamente um registro bastaria marca-lo de tal forma que os programa passariam a ignorá-los como se realmente não existisse. Uma opção para exclusão lógica de um registro seria criar um campo no registro (campo ***Ativo*** no exemplo acima) e considerar um valor para os registros válidos (S) e outro valor para registros excluídos (N). Desta forma, os programas para manipulação dos dados no arquivo (incluir, alterar, consultar, excluir e outros), vão ignorar

aqueles registros marcados como excluídos (Ativo = N), ou seja, serão considerados como se não existissem. No exemplo acima, registros cujo campo ***Ativo*** forem iguais a **S** são válidos, e iguais a **N** são excluídos.

➔ Para manipular cada campo de dado de um registro, deverão ser criadas nos programas variáveis de memória com os tipos de dados respectivos de cada dado (ex: String, float, int,...)

14.1 – Manipulação dos dados de um Registro em Java:

Como em java não há um tipo de dado para absorver o conceito de registros, para manipular os dados de cada registro no arquivo criaremos uma classe contendo variáveis de instância (atributos) para armazenar cada campo do registro do arquivo. A partir daí, cada instância da classe poderá armazenar os dados (compos ou atributos) de um registro.

Exemplo:

```
public class RegistroDemo {
  public static class Aluno {  // declarando a classe Aluno
    public char ativo;
    public String matricula;
    public String nomeAluno;
    public String dataNasc;
    public float mensalidade;
    public char sexo;
  }

  public static void main(String Args[]){
    Aluno aluno = new Aluno();  // criando a instância aluno da classe Aluno
    aluno.ativo = 'S';
    aluno.matricula ="054689";
    aluno.nomeAluno = "JOSE DA SILVA";
    aluno.dataNasc = "10/10/1980";
    aluno.mensalidade = (float)750;
    aluno.sexo = 'M';
    System.out.println("Matrícula do Aluno: " + aluno.matricula);
    System.out.println("Nome do Aluno.....: " + aluno.nomeAluno);
    System.out.println("Data de Nascimento: " + aluno.dataNasc);
    System.out.println("Valor Mensalidade.: " + aluno.mensalidade);
    System.out.println("Sexo..............: " + aluno.sexo);
  }
}
```

No exemplo acima, para trabalhar o conceito de registro, foi criada uma classe `Aluno` e declaradas variáveis públicas para armazenar os dados de cada campo do registro. Dentro do método `main`, foi criada uma instância da classe `Aluno` de nome `aluno`. A partiri deste ponto, a instância `aluno` armazenará os dados de um registro de aluno por vez.

14.2 – Classes e métodos para manipulação de Registros em Arquivos

As operações básicas de manipulação de registros em arquivos são a INCLUSÃO de novos registros no arquivo, a ALTERAÇÃO de dados dos registros existentes no arquivo, a CONSULTA de dados dos registros existentes e a EXCLUSÃO de registros.

No Java, as classes e métodos para manipulação de dados em arquivos estão contidas no pacote `java.io`. E todas as operações envolvendo arquivos que gerarem erros de acesso, como erros de leitura/gravação no disco, resultarão em exceções do tipo `IOExeption`(Input/Output Exeption). Estas exceções devem ser tratadas por meio do comando `try-catch` para evitar erros de execução caso as mesmas ocorrerem.

Exemplo:

```
try{
   RandomAccessFile arqAluno = new RandomAccessFile("C:\\LTPII\\ALUNOS.DAT", "rw");
   arqAluno.close();

}catch (IOException e) {
   System.out.println("Erro na abertura do arquivo - programa será finalizado");
   System.exit(0);  // cancela a excução do programa
}
```

A classe `RandomAccessFile` permite a gravação e a leitura de dados em arquivos de forma aleatória e disponibiliza vários métodos para manipulação destes dado.

No exemplo acima, um arquivo com nome `ALUNOS.DAT` será criado dentro da pasta `C:\LTPII`. Caso o arquivo já exista dentro desta pasta, ao invés de criado ele será apenas aberto para uso. Se o drive\diretório destino não for informado, o arquivo será gravado dentro da pasta (diretório) do projeto do Eclipse onde a classe (programa) foi criada. O parâmetro `"rw"` indica que o arquivo poderá ser utilizado para leitura ou gravação (read/write). Dentro do programa o arquivo `ALUNOS.DAT` será representado pela variável `arqAluno`.

Fechando um arquivo antes de encerrar o programa (`close()`):

Para evitar falha ou corrupção do arquivo de dados, antes de encerrar um programa ou o uso do arquivo, o mesmo deverá ser fechado com o método `close()`.
Exemplo:

```
arqAluno.close();
```

Descobrindo o tamanho de um arquivo em número de bytes (`length()`):

Exemplo:

```
System.out.println("Tamanho do arquivo ALUNOS.DAT: " + arqAluno.length() );
```

Posicionando o ponteiro em um byte do arquivo (`seek()`):

Exemplo:

```
arqAluno.seek(0);    // posiciona o ponteiro no início do arquivo
arqAluno.seek( arqAluno.length() );  // posiciona o ponteiro no final do arquivo (EOF)
```

Qual a posição atual do ponteiro em um byte do arquivo (`getFilePointer()`):

Exemplo:

```
System.out.println("Posição atual do cursor no arquivo: " + arqAluno.getFilePointer() );
```

Gravar dados no arquivo:

Para cada tipo de dados deve-se utilizar um método de gravação no arquivo:

variável_arquivo.writeByte(<valor>)	Grava um valor do tipo byte
variável_arquivo.writeInt(<valor>)	Grava um valor do tipo int
variável_arquivo.writeFloat(<valor>)	Grava um valor do tipo float
variável_arquivo.writeDouble(<valor>)	Grava um valor do tipo double
variável_arquivo.writeBoolean(<valor>)	Grava um valor do tipo boolean
variável_arquivo.writeChar(<valor>)	Grava um valor do tipo char
variável_arquivo.writeUTF(<valor>)	Grava um valor do tipo String

Exemplo, inserindo um registro no arquivo:

```
aluno.ativo       = 'S';
aluno.matricula   = "054689";
aluno.nomeAluno   = "JOSE DA SILVA";
aluno.dataNasc    = "10/10/1980";
aluno.mensalidade = 750;
aluno.sexo      = 'M';

try{
   RandomAccessFile arqAluno = new RandomAccessFile("G:\\LTPII\\ALUNOS.DAT", "rw");
   arqAluno.seek(arqAluno.length());  // posiciona ponteiro no fim do arquivo (EOF)
   arqAluno.writeChar(aluno.ativo);
   arqAluno.writeUTF(aluno.matricula);
   arqAluno.writeUTF(aluno.nomeAluno);
   arqAluno.writeUTF(aluno.dataNasc);
   arqAluno.writeFloat(aluno.mensalidade);
   arqAluno.writeChar(aluno.sexo);
   arqAluno.close();
   System.out.println("Dados gravados com sucesso !\n");

}catch (IOException e) {
   System.out.println("Erro na gravaçao do registro - programa será finalizado");
System.exit(0);
}
```

Ler dados de um arquivo:

Para cada tipo de dados deve-se utilizar um método de leitura no arquivo:

variável_arquivo.readByte(<valor>)	Lê um valor do tipo byte
variável_arquivo.readInt(<valor>)	Lê um valor do tipo int
variável_arquivo.readFloat(<valor>)	Lê um valor do tipo float
variável_arquivo.readDouble(<valor>)	Lê um valor do tipo double
variável_arquivo.readBoolean(<valor>)	Lê um valor do tipo boolean
variável_arquivo.readChar(<valor>)	Lê um valor do tipo char
variável_arquivo.readUTF(<valor>)	Lê um valor do tipo String

Exemplo, consultando um registro no arquivo:

```
System.out.println(" ***************  CONSULTA DE ALUNOS  ***************** ");
System.out.print("Digite a Matrícula do Aluno para consulta: ");
matriculaChave = leia.nextLine();

try{
   RandomAccessFile arqAluno = new RandomAccessFile("G:\\LTPII\\ALUNOS.DAT", "rw");
```

```
   while (true){
      aluno.ativo       = arqAluno.readChar();
      aluno.matricula   = arqAluno.readUTF();
      aluno.nomeAluno   = arqAluno.readUTF();
      aluno.dataNasc    = arqAluno.readUTF();
      aluno.mensalidade = arqAluno.readFloat();
      aluno.sexo        = arqAluno.readChar();
      if ( matriculaChave.equals(aluno.matricula) && aluno.ativo == 'S'){
         System.out.println("Nome do aluno.......: " + aluno.nomeAluno);
         System.out.println("Data de nascimento..: " + aluno.dataNasc);
         System.out.println("Valor da mensalidade: " + aluno.mensalidade);
         System.out.println("Sexo do aluno.......: " + aluno.sexo);
         break;
      }
    }
   arqAluno.close();

} catch (EOFException e){
   System.out.println("Esta matrícula não foi encontrada no arquivo !\n");

}catch (IOException e)  {
   System.out.println("Erro na abertura do arquivo - programa será finalizado");
   System.exit(0);
}
```

14.3 – Campo Chave (chave primária) de um registro

Para garantir que não existam registros repetidos e para localizar um registro em um arquivo deve-se utilizar o conceito de campo Chave. O campo Chave é representado por um dos campos de dados do registro que nunca contenham valores repetidos em registros diferentes do arquivo.

Quando não é possível utilizar apenas um campo de dados que identifique um registro sem que os valores deste campo, em registros diferentes, se repitam, pode-se criar um campo específico para este fim. Outra forma seriautilizar mais de um campo para identificar o registro, sendo que juntos, os valores destes campos são exclusivos para cada registro do arquivo.

Sendo assim, o campo Chave, por contervalor exclusivo entre diferentes registros, passa a ser o campo que identifica um registro no arquivo. Também é conhecido como chave primária de um arquivo de dados.

Exemplo:

```
public static class Aluno{
  public char ativo;
  public String matricula;
  public String nomeAluno;
  public String dtNasC;
  public float mensalidade;
  public char sexo;
}
```

Na estrutura de registro acima, qual dos campos poderia ser o campo chave ?

- `nomeAluno` => não poderia ser o campo chave porque é possível que exista mais de um aluno com o mesmo nome (homônimo), não sendo assim exclusivo para cada registro.

- `mensalidade`, `dataNasc` e `sexo`, também não poderiam ser o campo chave pelo mesmo motivo do `nomeAluno`

- `matricula` => a matrícula é o campo adequado para ser a chave primária, porque ela existe no registro do aluno exatamente para identificá-lo e diferenciá-lo de outros alunos, sendo assim, cada aluno terá uma matrícula diferente.

Então usaremos a `matricula` com campo chave para localizar um registro no arquivo e não será permitida a inclusão alunos que contenha a mesma `matricula`, ou seja, o programa que inclui registros no arquivo deverá garantir que só sejam incluídos alunos com o números de `matricula` diferentes.

14.4 – Exemplos de programas

14.4.1 - Exemplo 1:

Programa para INCLUSÃO de dados em um registro de Alunos gravado em um arquivo em disco de nome ALUNOS.DAT

```
import java.io.*;
import java.util.*;
public class Inclusao {
  public static class Aluno {
    public char    ativo;
    public String  matricula;
    public String  nomeAluno;
    public String  dtNasc;
    public float   mensalidade;
    public char    sexo;
  }
  public static void main(String[] args) {
    Aluno aluno = new Aluno();
    RandomAccessFile arquivo;
    Scanner leia = new Scanner(System.in);
    boolean encontrou;
    String matriculaChave;
    char confirmacao;
    do {
      do {
          System.out.println(" ***************  INCLUSAO DE ALUNOS  ***************** ");
          System.out.print("Digite a Matrícula do Aluno( FIM para encerrar): ");
          matriculaChave = leia.nextLine();
          if (matriculaChave.equals("FIM")) {
             break;
          }
          encontrou = false;
          try {
             arquivo = new RandomAccessFile("ALUNOS.DAT", "rw");
             while (true) {
                 aluno.ativo       = arquivo.readChar();
                 aluno.matricula   = arquivo.readUTF();
                 aluno.nomeAluno   = arquivo.readUTF();
                 aluno.dtNasc      = arquivo.readUTF();
                 aluno.mensalidade = arquivo.readFloat();
                 aluno.sexo        = arquivo.readChar();
```

```
                if ( matriculaChave.equals(aluno.matricula) && aluno.ativo == 'S') {
                      System.out.println("Matrícula já cadastrada, digite outro valor\n");
                      encontrou = true;
                      break;
                }
             }
                arquivo.close();
          }catch (EOFException e) {
                encontrou = false;
          }catch (IOException e) {
                System.out.println("Erro na abertura do arquivo - programa será finalizado");
                System.exit(0);
          }
       }while (encontrou);

       if (matriculaChave.equals("FIM")) {
            System.out.println("\n ************  PROGRAMA ENCERRADO  ************** \n");
            break;
       }
       aluno.ativo = 'S';
       aluno.matricula = matriculaChave;
       System.out.print("Digite o nome do aluno.........................: ");
       aluno.nomeAluno = leia.nextLine();
       System.out.print("Digite a data de nascimento (DD/MM/AAAA).......: ");
       aluno.dtNasc = leia.nextLine();
       System.out.print("Digite o valor da mensalidade..................: ");
       aluno.mensalidade = leia.nextFloat();
       System.out.print("Digite o Sexo do aluno (M/F)...................: ");
       aluno.sexo = leia.next().charAt(0);
       do {
            System.out.print("\nConfirma a gravação dos dados (S/N) ? ");
            confirmacao = leia.next().charAt(0);
            if (confirmacao == 'S') {
               try {
                  arquivo = new RandomAccessFile("ALUNOS.DAT", "rw");
                  arquivo.seek(arquivo.length());  // posiciona no final do arquivo (EOF)
                  arquivo.writeChar(aluno.ativo);
                  arquivo.writeUTF(aluno.matricula);
                  arquivo.writeUTF(aluno.nomeAluno);
                  arquivo.writeUTF(aluno.dtNasc);
                  arquivo.writeFloat(aluno.mensalidade);
                  arquivo.writeChar(aluno.sexo);
                  arquivo.close();
                  System.out.println("Dados gravados com sucesso !\n");
               }catch (IOException e) {
                  System.out.println("Erro na gravaçao do registro - programa será finalizado");
                        System.exit(0);
                }
            }
          }while (confirmacao != 'S' && confirmacao != 'N');
          leia.nextLine();
    }while ( ! aluno.matricula.equals("FIM"));
    leia.close();
  }
}
```

14.4.2 - Exemplo 2:

Programa para ALTERAÇÃO de dados em um registro de Alunos gravado em um arquivo em disco de nome ALUNOS.DAT

```
import java.io.*;
import java.util.*;
public class Alteracao {
   public static class Aluno    {
     public char ativo;
     public String matricula;
     public String nomeAluno;
     public String dtNasc;
     public float mensalidade;
     public char sexo;
   }
   public static void main(String[] args) {
     Aluno aluno = new Aluno();
     RandomAccessFile arquivo;
     Scanner leia = new Scanner(System.in);
     boolean encontrou;
     String matriculaChave;
     char confirmacao;
     long posicaoRegistro = 0;
     byte opcao;

     do{
        do{
            System.out.println(" ***************  ALTERAÇÃO DE ALUNOS  ***************** ");
            System.out.print("Digite a Matrícula do Aluno ( FIM para encerrar ): ");
            matriculaChave = leia.nextLine();
            if (matriculaChave.equals("FIM")) {
               break;
            }
            encontrou = false;
            try {
                arquivo = new RandomAccessFile("ALUNOS.DAT", "rw");
                while (true) {
                  // guarda a posição inicial do  registro a ser alterado
                  posicaoRegistro = arquivo.getFilePointer();
                  aluno.ativo        = arquivo.readChar();
                  aluno.matricula    = arquivo.readUTF();
                  aluno.nomeAluno    = arquivo.readUTF();
                  aluno.dtNasc       = arquivo.readUTF();
                  aluno.mensalidade  = arquivo.readFloat();
                  aluno.sexo         = arquivo.readChar();
                  if ( matriculaChave.equals(aluno.matricula) && aluno.ativo == 'S') {
                        encontrou = true;
                        break;
                  }
                }
                arquivo.close();
            } catch (EOFException e) {
                encontrou = false;
                System.out.println("Esta matrícula não foi encontrada no arquivo !\n");
            } catch (IOException e) {
                System.out.println("Erro na abertura do arquivo - programa será finalizado");
                System.exit(0);
            }
        }while ( ! encontrou);

        if (matriculaChave.equals("FIM")) {
                  System.out.println("\n ************  PROGRAMA ENCERRADO  ************** \n");
                  break;
        }

        aluno.ativo = 'S';
```

```
        aluno.matricula = matriculaChave;

        do{
             System.out.println("[ 1 ] Nome do Aluno.............: " + aluno.nomeAluno);
             System.out.println("[ 2 ] Data de nascimento .......: " + aluno.dtNasc);
             System.out.println("[ 3 ] Valor da mensalidade......: " + aluno.mensalidade);
             System.out.println("[ 4 ] sexo do Aluno.............: " + aluno.sexo);

             do{
                System.out.println("Digite o número do campo que deseja alterar (0 para finalizar
as alterações): ");
                opcao = leia.nextByte();
             }while (opcao < 0 || opcao > 4);

             switch (opcao) {
                  case 1:
                         leia.nextLine();
                         System.out.print  ("Digite o NOVO NOME do Aluno..................: ");
                         aluno.nomeAluno = leia.nextLine();
                         break;
                  case 2:
                         leia.nextLine();
                         System.out.print  ("Digite a NOVA DATA de Nascimento (DD/MM/AAAA): ");
                         aluno.dtNasc = leia.nextLine();
                         break;
                  case 3:
                         System.out.print  ("Digite o NOVO VALOR da mensalidade...........: ");
                         aluno.mensalidade = leia.nextFloat();
                         break;
                  case 4:
                         System.out.print  ("Digite o NOVO sexo do Aluno (M/F)............: ");
                         aluno.sexo = leia.next().charAt(0);
                         break;
              }
              System.out.println();
        }while (opcao != 0);

        do{
            System.out.print("Confirma as Alterações (S/N) ? ");
            confirmacao = leia.next().charAt(0);
            if (confirmacao == 'S') {
              try {
                   arquivo = new RandomAccessFile("ALUNOS.DAT", "rw");
                   // desativando o registro atual
                   arquivo.seek(posicaoRegistro);
                   // gravando N por cima do atual valor S contido no campo ATIVO
                   arquivo.writeChar('N');
                   // gravando um novo registro contendo os novos dados das alterados
                   arquivo.seek(arquivo.length());  // posiciona final do arquivo (EOF)
                   arquivo.writeChar(aluno.ativo);
                   arquivo.writeUTF(aluno.matricula);
                   arquivo.writeUTF(aluno.nomeAluno);
                   arquivo.writeUTF(aluno.dtNasc);
                   arquivo.writeFloat(aluno.mensalidade);
                   arquivo.writeChar(aluno.sexo);
                   arquivo.close();
                   System.out.println("Dados alterados com sucesso !\n");
               } catch (IOException e) {
                   System.out.println("Erro na gravaçao do registro - programa será finalizado");
                   System.exit(0);
               }
            }
```

```
            System.out.println();
          }while (confirmacao != 'S' && confirmacao != 'N');

          leia.nextLine();
      }while ( ! aluno.matricula.equals("FIM"));

      leia.close();
   }
}
```

14.4.3 - Exemplo 3:

Programa para CONSULTA de dados em um registro de Alunos gravado em um arquivo em disco de nome ALUNOS.DAT

```
import java.io.*;
import java.util.*;
public class Consulta {
   public static class Aluno {
     public char ativo;
     public String matricula;
     public String nomeAluno;
     public String dtNasc;
     public float mensalidade;
     public char sexo;
   }
   public static void main(String[] args) {
     Aluno aluno = new Aluno();
     RandomAccessFile arqAluno;
     Scanner leia = new Scanner(System.in);
     byte opcao;
     String matriculaChave;
     char sexoAux;
     long posicaoRegistro;
     boolean encontrou;
     do {
           System.out.println(" ***************  CONSULTA DE ALUNOS  ***************** ");
           System.out.println(" [1] CONSULTAR APENAS 1 ALUNO ");
           System.out.println(" [2] LISTA DE TODOS OS ALUNOS ");
           System.out.println(" [3] LISTA SOMENTE SEXO MASCULINO OU FEMININO ");
           System.out.println(" [0] SAIR");
           do {
                 System.out.print("\nDigite a opção desejada: ");
                 opcao = leia.nextByte();
                 if (opcao < 0 || opcao > 3) {
                 System.out.println("opcao Inválida, digite novamente.\n");
                 }
           }while (opcao < 0 || opcao > 3);

           switch (opcao) {
              case 0:
                 System.out.println("\n ************  PROGRAMA ENCERRADO  ************* \n");
                 break;

              case 1:  // consulta de uma única matrícula
                 leia.nextLine();  // limpa buffer de memória
                 System.out.print("Digite a Matrícula do Aluno: ");
                 matriculaChave = leia.nextLine();
                 encontrou = false;
```

```
    try {
        arqAluno = new RandomAccessFile("ALUNOS.DAT", "rw");
        while (true) {
          // guardar a posição inicial do registro a ser alterado
          posicaoRegistro = arqAluno.getFilePointer();
          aluno.ativo               = arqAluno.readChar();
          aluno.matricula     = arqAluno.readUTF();
          aluno.nomeAluno     = arqAluno.readUTF();
          aluno.dtNasc        = arqAluno.readUTF();
          aluno.mensalidade   = arqAluno.readFloat();
          aluno.sexo          = arqAluno.readChar();
          if ( matriculaChave.equals(aluno.matricula) && aluno.ativo == 'S') {
            encontrou = true;
            imprimirCabecalho();
            imprimirAluno(aluno);
            System.out.println("\n FIM DE RELATÓRIO - ENTER para continuar...\n");
            leia.nextLine();
            break;
          }
        }
        arqAluno.close();
    } catch (EOFException e) {
        encontrou = false;
        System.out.println("Esta matrícula não foi encontrada no arquivo !\n");
    } catch (IOException e) {
        System.out.println("Erro na abertura arquivo - programa será finalizado");
        System.exit(0);
    }
    break;

case 2:  // imprime todos os alunos
    try {
        arqAluno = new RandomAccessFile("ALUNOS.DAT" , "rw");
        imprimirCabecalho();
        while (true) {
          aluno.ativo           = arqAluno.readChar();
          aluno.matricula    = arqAluno.readUTF();
          aluno.nomeAluno    = arqAluno.readUTF();
          aluno.dtNasc       = arqAluno.readUTF();
          aluno.mensalidade  = arqAluno.readFloat();
          aluno.sexo         = arqAluno.readChar();
          if ( aluno.ativo == 'S') {
                imprimirAluno(aluno);
          }
        }
    } catch (EOFException e) {
        System.out.println("\n FIM DE RELATÓRIO - ENTER para continuar...\n");
        leia.nextLine();
    } catch (IOException e) {
        System.out.println("Erro na abertura arquivo - programa será finalizado");
        System.exit(0);
    }
    break;

case 3:  // imprime alunos do sexo desejado
    do {
       System.out.print("Digite o Sexo desejado (M/F): ");
       sexoAux = leia.next().charAt(0);
       if (sexoAux != 'F' && sexoAux != 'M') {
          System.out.println("Sexo Inválido, digite M ou F");
       }
    }while (sexoAux != 'F' && sexoAux != 'M');
```

```
                try {
                   arqAluno = new RandomAccessFile("ALUNOS.DAT", "rw");
                   imprimirCabecalho();
                   while (true) {
                       aluno.ativo           = arqAluno.readChar();
                       aluno.matricula    = arqAluno.readUTF();
                       aluno.nomeAluno    = arqAluno.readUTF();
                       aluno.dtNasc       = arqAluno.readUTF();
                       aluno.mensalidade = arqAluno.readFloat();
                       aluno.sexo         = arqAluno.readChar();
                       if ( sexoAux == aluno.sexo && aluno.ativo == 'S') {
                             imprimirAluno(aluno);
                       }
                   }
                } catch (EOFException e) {
                   System.out.println("\n FIM DE RELATÓRIO - ENTER para continuar...\n");
                   leia.nextLine();
                   matriculaChave = leia.nextLine();
                } catch (IOException e) {
                   System.out.println("Erro na abertura arquivo - programa será finalizado");
                   System.exit(0);
                }

           }

     } while ( opcao != 0 );
   }

   public static void imprimirCabecalho () {
     System.out.println("-MATRÍCULA-  -------- NOME ALUNO ----------  --DATA NASC--  -
Mensalidade-  -sexo- ");
   }

   public static void imprimirAluno (Aluno aluno) {
           System.out.println(formatarString(aluno.matricula, 11 ) + "  " +
                       formatarString(aluno.nomeAluno , 30) + "  " +
                       formatarString(aluno.dtNasc , 13) + "  " +
                       formatarString( String.valueOf(aluno.mensalidade) , 13 ) + "  " +
                       formatarString( Character.toString(aluno.sexo) , 6 )   );
     }

   public static String formatarString (String texto, int tamanho) {
     // retorna uma string com o número de caracteres passado como parâmetro em TAMANHO
     if (texto.length() > tamanho) {
           texto = texto.substring(0,tamanho);
     }else{
           while (texto.length() < tamanho) {
                 texto = texto + " ";
           }
     }
     return texto;
   }

}
```

14.4.4 - Exemplo 4:

Programa para EXCLUSÃO de dados em um registro de Alunos gravado em um arquivo em disco de nome ALUNOS.DAT

Obs: como a exclusão física de registros é uma operação complexa e demorada, normalmente o que se faz é uma exclusão lógica dos registros. No programa abaixo, a exclusão lógica se trata apenas de alterar o valor do campo `ativo`para o valor `'N'`. Sendo assim, nos programas de inclusão, alteração, consulta e no próprio de exclusão, toda vez que um registro tiver o valor do campo `ativo = 'N'`, este registro será ignorado e tratado como se não existisse, ou seja, está excluído !

```
import java.io.*;
import java.util.*;

public class Exclusao {
   public static class Aluno  {
     public char ativo;
     public String matricula;
     public String nomeAluno;
     public String dtNasc;
     public float mensalidade;
     public char sexo;
   }
   public static void main(String[] args) {
     Aluno aluno = new Aluno();
     RandomAccessFile arqAluno;
     Scanner leia = new Scanner(System.in);
     boolean encontrou;
     String matriculaChave;
     char confirmacao;
     long posicaoRegistro = 0;
     do {
        do {
            System.out.println(" ***************  EXCLUSÃO DE ALUNOS  ***************** ");
            System.out.print("Digite a Matrícula do Aluno para excluir ( FIM para encerrar ): ");
            matriculaChave = leia.nextLine();

            if (matriculaChave.equals("FIM")) {
               break;
            }

            encontrou = false;
            try {
               arqAluno = new RandomAccessFile("ALUNOS.DAT", "rw");
               while (true) {
                   posicaoRegistro = arqAluno.getFilePointer();  // guarda a posição inicial do
                                                                 registro a ser excluído
                   aluno.ativo         = arqAluno.readChar();
                   aluno.matricula   = arqAluno.readUTF();
                   aluno.nomeAluno  = arqAluno.readUTF();
                   aluno.dtNasc     = arqAluno.readUTF();
                   aluno.mensalidade = arqAluno.readFloat();
                   aluno.sexo        = arqAluno.readChar();
                   if ( matriculaChave.equals(aluno.matricula) && aluno.ativo == 'S') {
                          encontrou = true;
                          break;
                   }
               }
               arqAluno.close();
            } catch (EOFException e) {
                   encontrou = false;
                   System.out.println("Esta matrícula não foi encontrada no arquivo !\n");
            } catch (IOException e) {
                   System.out.println("Erro na abertura do arquivo - programa será finalizado");
                   System.exit(0);
```

```
            }
        } while ( ! encontrou);

        if (matriculaChave.equals("FIM")) {
            System.out.println("\n ************  PROGRAMA ENCERRADO  ************** \n");
            break;
        }
        aluno.ativo = 'N';  // desativar o registro => exclusão
        System.out.println("Nome do aluno.......: " + aluno.nomeAluno);
        System.out.println("Data de nascimento..: " + aluno.dtNasc);
        System.out.println("Valor da mensalidade: " + aluno.mensalidade);
        System.out.println("Sexo do aluno.......: " + aluno.sexo);
        System.out.println();
        do {
            System.out.print("\nConfirma a exclusão deste aluno (S/N) ? ");
            confirmacao = leia.next().charAt(0);
            if (confirmacao == 'S') {
               try {
                     arqAluno = new RandomAccessFile("ALUNOS.DAT", "rw");
                     // desativando o registro => exclusão
                     arqAluno.seek(posicaoRegistro);
                     arqAluno.writeChar(aluno.ativo);
                     arqAluno.close();
                     System.out.println("Aluno excluído com sucesso !\n");
                } catch (IOException e) {
                   System.out.println("Erro na gravaçao do registro - programa será finalizado");
                   System.exit(0);
                }
             }
        } while (confirmacao != 'S' && confirmacao != 'N');
        leia.nextLine();
     } while ( ! aluno.matricula.equals("FIM"));
     leia.close();
   }
}
```

EXERCÍCIOS (parte 11):

Questão 11.1 - Fazer um programa em Java para incluir dados em um arquivo de Clientes de uma Empresa, de acordo com a estrutura de registro abaixo:

```
ativo             -> char    - gravar 'S' na inclusão e 'N' na exclusão
codCliente        -> int     - código do cliente
nomeCliente       -> String  - nome do cliente
vlrCompra         -> float   - valor da compra
anoPrimeiraCompra -> int     - ano que o cliente fez a primeira compra
emDia             -> boolean - se o cliente está em dia com o pagamento
```

Obs: - utilizar o campo CODCLI como campo chave primária.

CONSISTÊNCIAS:

- o nome do cliente deve ter no mínimo 10 caracteres.

- o código do cliente deve ser número inteiro e maior que (zero).

- o valor da compra deve ser maior que zero.

- o ano da primeira compra dever ser menor ou igual a 2013.

- `emDia` : por ser um campo do tipo boolean (lógico), no arquivo este campo deve ser preenchido com o valor TRUE ou FALSE. Entretanto, para exibir o valor na tela o programa deverá exibir S ou N, e o usuário deverá digitar também o valor S ou N para este campo. Para isto, o programa deverá utilizaruma variável auxiliar do tipo char e solicitar ao usuário que responda a pergunta: (*"Cliente está em dia (S/N)?"*). Se a resposta for `'S'`, atribuir TRUE ao campo `emDia`, caso contrário, atribuir o valor FALSE

- Quando o `codCliente` for digitado, verificar no arquivo se já existe algum outro registro que já possua o `codCliente` informado e com `ativo == 'S'` (registro não excluído). Caso existir, mostrar uma mensagem de erro ("*Cliente já cadastrado !*"), e não permitir a inclusão.

Questão 11.2 - Fazer os programas para alterar, consultar e excluir dados dos registros no arquivo do exercício anterior.