UNIVERSIDADE
FUMEC/FACE
Faculdade de Ciências Empresariais

LTP 2
Java

CIÊNCIA DA COMPUTAÇÃO

PROF. AIR RABELO

Agosto 2018

1 - Introdução a Programação

1.1 - Algoritmo

É a descrição seqüencial de ações a serem executados para o cumprimentode 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 - 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.

1.3 - 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.

1.4 – Programa 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.

1.5 – Programa 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.

1.6 – 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.

1.7 – A tabela ASCII

ASCII (*American Standard Code for Information Interchange)*, em português: "Código Padrão Americano para o Intercâmbio de Informação". É umacodificação de caracteres de sete bits (2^7 = 128 códigos) ou 8 bits - ASCII estendida (2^8 = 256 códigos) baseada no alfabeto inglês. Cada sequencia de códigos na tabela ASCII corresponde a um caractere, comumente representados pelos 8 bits (equivalente a um byte). Os códigos ASCII representam texto em computadores. Desenvolvida a partir de 1960, grande parte das codificações de caracteres modernas a herdaram como base.

O quadro abaixo mostra uma parte da tabela ASCII com os caracteres mais utilizados em programação:

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	~

2 - A Linguagem JAVA

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.

3 - Estrutura de um programa em Java:

Umprograma em javaé estruturado 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) {//declaração do 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.

4 – 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:0

```
/* comentário 1
comentário 2  */

// comentário 1
// comentário 2
```

5 – 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, ...).

5.1 – Nomes de Variáveis e outros tipos

Os nomes dados a 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 ouos caracteres*underline(_)* e $
- os nomes devem ser simples e descritivos e podem conter qualquer quantidade de caracter.

Nomes segundo os Padrões de codificação Java definidos pela Sun:

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`

Classes e tipos de dados: 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`.

Constantes: devem ser escritas com letras maiúsculas, exemplo: `PI` , `VELOCIDADE_LUZ`

5.2 – Tipos de Variáveis

Estudaremos variáveis de 3 classes:

- 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.

5.2.1 - Tipos de Dados

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`)

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

5.3 – Declaração de Variáveis no Java

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.

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 locais para aquela classe e globais para os métodos existentes dentro daquela classe, ou seja, são válidas em todos os métodos criados 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;
     }
}
```

5.4 – 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 (“).

Exemplo:

```
idade = 25;                                  // tipo int
salario = 2786.50;                           // tipo float
nomeAluno = "Marcelo dos Santos";            // tipo String
sexo = 'F';                                  // tipo char
alunoFoiAprovado = true;                     // tipo boolean
```

6 – Linha de Comando no Java

Uma linha de comando no Java pode ocupar uma ou mais linhas no programa 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, a não ser que os motivos sejam bem claros.

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.

7 – Operadores

7.1 - Operadores Aritméticos

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

Operador	O que representa	Exemplo de uso
+	**Soma**	`b + c`
-	**Subtração**	`b – a`
*	**Multiplicação**	`x * y`
/	**Divisão**	`x / y`
%	**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 (equivale a dois comandos)	`b = --a`	`// mesmo que a = a - 1` `// em seguida b = a`
= --	Atribuição com decremento de 1 (equivale a dois comandos)	`b = a--`	`// mesmo que b = a` `// em seguida a = a - 1`

7.1.1 - Divisão entre valores Inteiros

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` .

7.2 - Operadores Relacionais

Os operadores relacionais são utilizados para efetuar a comparação entre dadosde mesmo tipo. São eles:

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 a = 1, b = 3 e c = 5

```
c == a   // false
a > b    // false
b <= c   // true
a != b   // true
```

7.3 - 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)

Exemplos: considere as variáveis a = 1 , b = 3 e c = 5

```
((a > b)&&(b <= c))       // FALSE e TRUE  =>   FALSE
((a != b) || (c < b))     // TRUE ou FALSE =>   TRUE
(!(b <= c))               // 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 para facilitar o entendimento das prioridades da execução.

7.4 -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

7.5 - Prioridade das Operações

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 → ! , && , ||

7.6 - Operador de Concatenação

O operador de concatenação (**+**) efetua a junção de duas variáveis ou constantes do tipo `String`ou `char`.

Exemplo:

```
public class ExemploConcatenacao {
     public static void main(String[] args) {
          String nome;
          String sobreNome;
          String nomeCompleto;
          // Suponhamos o nome Josias Santos
          nome = "Josias";
          sobreNome = "Santos";
          nomeCompleto = nome + ' ' + sobreNome;
          System.out.println( nomeCompleto );    // Josias Santos
          nomeCompleto = "Jose" + ' ' + "Maria";
          System.out.println( nomeCompleto );    // Jose Maria
     }
}
```

8 – Comandos de Entrada e Saída de dados

8.1 - Saída de dados

Os principais comandos destinadas a exibir os dados no vídeo ou em uma impressora são:
`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 aclasse`DecimalFormat` que está contida no pacote de classes `text`, que deverá ser importado no início do programa `(import java.text.*;)`.

8.2 – 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:

```
import java.util.*;
public class ExemploEntrada {
  public static void main(String[] args) {
    // declaração das variáveis
    String nome;
    char sexo;
    float salario;
    byte idade;
    // entrada dos dados
    Scanner leia;          // declara a variável leiapara ser utilizada na classe Scanner
```

```
    leia = new Scanner(System.in);   /* inicializa a variável leiapara receber os
                                        valores de entrada da classe Scanner*/
    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

    // 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);
  }
}
```

Programa exemplo:

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.*;
import java.text.*;
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 - cálculos
          novoSal = salario + vlrAumento;

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

          // Exemplo da saída de dados utilizando formatação de casas decimais
```

```
            DecimalFormat decimal;      /* declaração da variável DECIMAL para ser
                                           utilizada na classe DecimalFormat  */
            /*abaixo, inicialização da variável DECIMAL com formatação para impressão de números reais
            com duas casas decimais e separador de milhar */
            decimal = new DecimalFormat("#,##0.00");
           System.out.print("Novo salário:" + decimal.format(novoSal));
      }
}
```

9 – Comando de alternativa – IF

9.1 – Alternativa Simples

Sintaxe:

Um comando por alternativa	Mais de um comando por alternativa

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

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

Exemplo1:

```
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 ! ");
}
```

Exemplo2:

```
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;
}
```

9.2 – Alternativa Composta

Sintaxe:

Um comando por alternativa	Mais de um comando por alternativa	Mais de uma alternativa (else if)

`if ( condição ) {`
`  comando1;`
`} else {`
`  comando2;`
`}`
	`if ( condição ) {`
	`  comando1;`
	`  comando2;`
	`} else{`
	`  comando3;`
	`  comando4;`
	`}`
		`if ( condição1 ) {`
		`  comando1;`
		`  comando2;`
		`} else if ( condição2 ) {`
		`  comando3;`
		`} else if ( condição3 ) {`
		`  Comando4;`
		`  Comando5;`
		`} else {`
		`  Comando6;`
		`}`

Exemplo1:

```
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 ! ");
}
```

Exemplo2:

```
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: ");
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 ! ");
}
```

Exemplo3:

```
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;
}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 ! ");
     idade = idade - 5;
}
```

10 - 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 1):

Exercício 1.1 - 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 1.2 – 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;
- eqüilátero, se os 3 lados forem iguais;
- isósceles, se 2 lados forem iguais e 1 diferente;
- escaleno, se os 3 lados forem diferentes;

Exercício 1.3 – 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

10 – Comandos de repetição - *loop*

10.1 – while (enquanto...faça)

Caracteriza-se por verificar uma condiçãoantes 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:

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

Exemplo 2:

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

10.2 – 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:

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

10.3 – 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;
}
                ou

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

Obs: quando existir apenas uma linha de comando dentro da estrutura de repetição `for`, 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.

Exemplo 1:

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

```
  altura = altura + 2;
  largura--;
}
```

Exemplo 2:

```
byte i;
for (i = 1; i <= 9; i = i + 2) {// i será incrementado de 2 em 2
  System.out.println("O valor de I é: " + i );
}
```

Para se executar o ***for*** com passo negativo, ou seja, ao invés do valor da variável <var1> ser incrementado de 1 ou outro valorem cada execução do loop, ele ser decrementado de 1 ou outro valor.

Exemplo:

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

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

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

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

OBS: Como interromper uma estrutura de repetição em Java ?

Exemplo:

O comando `break` interrompe qualquer uma das estruturas de repetição, fazendo com que a execução do programa prossiga a partir da primeira linha após o final do comando de repetição

```
for (i=1; i<=5; i++) {
  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);
  .
  .
  .
}
```

Neste exemplo acima, apesar do comando `for` utilizar o contador `i` que varia de `1` a `5`, se o valor digitado na variável `idade` for 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`.

EXERCÍCIOS (parte 2):

Exercício 2.1- 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 2.2– 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 2.3- 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

11 – Métodos da linguagem Java - Parte 1

11.1 –COMPARAR SE DUAS STRINGS SÃO IGUAS

```
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");
}
```

11.2–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.

12 - VETORES e MATRIZES:

Ao se utilizar variáveis, pode-se armazenar apenas um valor por vez. Considere um programa que precisa armazenar as notas de 5 provas realizadas por um aluno. Uma opção seria criar cinco variáveis para armazenar as notas. Por exemplo:

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

Existem estruturas que permitem agrupar várias informações dentro de uma mesma variável. Estas estruturas são chamadas de **vetores** e **matrizes.**

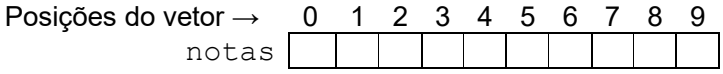
12.1 – Vetores

Este tipo de estrutura é também chamado de **matriz unidimensional**. Um vetoré declarado com seu nome, tamanho e seu 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.

Exemplo 1:

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

Ou seja, o vetor `notas`começa na posição 0 e termina na posição 9, total de 10 itens:



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

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

Exemplo 2:

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

A atribuição ou a exibição de valores contidos nos vetores é feita indicando a posição do vetor entre colchetes. Exemplo:Atribuindo valores ao vetor de notas do aluno:

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

O nome é o mesmo, o que muda é a informação indicada dentro doscolchetes, ou seja, o índice ou posição do vetor, que representa a posição dentro do vetor onde ovalor está armazenado. Para o exemplo anterior teríamos:

notas

0	1	2	3	4
5.2	**8.0**	**9.2**	**7.5**	**8.3**

=> índice ou posição no vetor
=> conteúdo de cada posição

12.2 – Matrizes

Usando mesmo exemplo do início desta página, vamos supor que agora será necessário armazenar 4 notas dos mesmos 5 alunos, ao invés de apenas uma. Uma opção seria criar cinco vetores, um para cada 4 notas de cada aluno. Entretanto, esta não é a melhor solução já que seria necessário criar uma estrutura de controle para cada vetor. A melhor opção seria criar uma única matriz de duas dimensões para armazenar cada uma das 4 notas dos 5 alunos.

Para declarar matrizes em java, é da mesma forma que a 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. No caso de uma matriz de 2 dimensões, a primeira dimensão indica o número de linhas e a segunda o número de colunas.

Para o caso das 4 notas dos 5 alunos a declaração da matriz seria:

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

ou

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

Neste caso, as linhas representam os alunos, e as colunas representas as notas:

notas

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 notas

Índice das linhas => 5 alunos

Exemplo de programa:

```
import java.util.*;
public class Exemplo {
  public static void main(String[] args){
    String nomes[] = new String[50];
    byte idades[] = new byte[30];
    int quantAlunosPorNota[] = new int[100];
    String nomeAlunos[][] = new String[8][7];
    int x;
    idades[1] = 74;  // atribuição do valor 74 para a posição 1 do vetor
    Scanner leia = new Scanner(System.in);
    for (x = 0; x <= 49; x++){
       System.out.print("Digite o Nome " + x + ": ");
       nomes[x] = leia.nextLine();
    }
    System.out.println(nomes);// Comando errado por não ter sido informada a
                              // posição desejada para impressão do vetor NOMES
    System.out.println(nomes[10]);  // comando correto
  }
}
```

12.3 - 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 3):

Exercício 3.1 – 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.

13 – 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.

13.1 - 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.

13.2 - 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.

13.3 - 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 métodoé 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;
    subtrai();
    System.out.println("Valor de X:" + x);       // X => 3
    System.out.println("Valor de Y:" + y);       // Y => 9
  }

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

13.4 - 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;

    subtrai(x,y);

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

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

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

Neste exemplo o método SUBTRAI 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 (`subtrai(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 (`subtrai(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 `subtrai`, 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***.

13.5– 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;
```

```
    subtrai(x,y);
    System.out.println("Valor de X:" + x[0]);    // X => 3
    System.out.println("Valor de Y:" + y[0]);    // Y => 9
    subtrai(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 subtrai (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 `subtrai(x,y)` quanto na segunda `subtrai(w,q)` os vetores `x,y,w` e`q`são 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 `subtrai(x,y)` e `subtrai(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`.

13.6 – 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.println("o fatorial de " + valor + " é:" + fatorial(valor));
  }
  public static long fatorial(int n){
    long fat;
    int i;
    fat = 1;
    for ( i = 1; i <= n ; i++ ) {
      fat = fat * i;
    }
    return fat;
  }
}
```

Neste exemplo, o método `fatorial`, 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.

13.7 – 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 `fatorial`.

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

Exemplo 5 - primeira parte - Rotinas.java

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

  public static int fatorial( 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.fatorial(valor));
  }
}
```

EXERCÍCIOS (parte 4):

Exercicio 4.1 - 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 4.2 - 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	Média Salários
XXXXXXXXXXXXXXXX	999.99	999.99
XXXXXXXXXXXXXXXX	999.99	999.99
XXXXXXXXXXXXXXXX	999.99	999.99
XXXXXXXXXXXXXXXX	999.99	999.99

Total pago com salários: 9,999.99

Exercício 4.3 - 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
```

14 – Métodos da linguagemJava - Parte 2

14.1 - 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
```

14.2 – 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*.

14.2.1–RETORNAR PARTE DE UMA STRING

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

O método substringretorna 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
```

14.2.2 – 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
```

14.2.3 – 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
```

14.2.4–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"`.

14.2.5–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'`.

14.2.6 – 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);
```

14.2.7 – 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.

14.3 - Outros métodos de manipulação de Strings:

14.3.1 – 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.
```

14.3.2 – 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.
```

EXERCÍCIOS (parte 5):

Exercício 5.1 – 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 5.2 – 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 5.3 - 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 5.4 - 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.

15 - Métodos para manipular Strings - Parte 3

15.1 –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
```

15.2 –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
```

15.3 –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);
```

15.4 – 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 !");
}
```

15.5 – 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");
}
```

15.6 – 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);
```

15.7 – 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 `"/"`

15.8 – 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 `"/"`

15.9 – 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 6):

Exercício 6.1 – 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 6.2 – 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.

15 – 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,...)

15.1 – 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.

15.2 –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);
}
```

15.3 – 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.

15.4 – Exemplos de programas

15.4.1 - 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();
    }
 }
```

15.4.2 - 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();
    }
}
```

15.4.3 - 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;
   }

}
```

15.4.4 - 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 7):

Questão 7.1 - 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 7.2 - Fazer os programas para alterar, consultar e excluir dados dos registros no arquivo do exercício anterior.