



Pós-Graduação a Distância

Lógica de Programação

Prof. Maurício Duarte
Profa. Elisamara de Oliveira

Sumário

Apresentação	4
Módulo 1: Introdução ao Java	5
Aula 1: Primeira aplicação em Java	5
Conceitos preliminares – ambiente de programação	5
Primeiros passos	9
Entendendo o primeiro programa	11
Exercício	12
Aula 2: Tipos de dados, variáveis e operadores	12
Tipos de dados primitivos e declaração de variáveis	12
Operadores	14
Exercícios	16
Aula 3: Entrada e saída de dados	17
Entrada de dados em Java	17
Saída de dados em Java	18
Exercícios	20
Exercícios do Módulo 1	20
Módulo 2: Estruturas de controle	22
Aula 4: Estruturas de seleção ou decisão	22
Estrutura de seleção composta if ... else	23
Exercícios	25
Estrutura de seleção múltipla switch... case	25
Aula 5: Estruturas de repetição	26
Repetição com teste no início	27
Exercícios	28
Repetição com teste no final	28
Exercício	29
Repetição com variável de controle	29
Exercícios	31
Comandos break e continue	32
Exercício	33
Exercícios do Módulo 2	33
Módulo 3: Estruturas de dados básicas	34
Aula 6: Vetores (<i>arrays</i>)	34
Declaração de vetores	34
Inicialização de vetores	35
Manipulação dos elementos de um vetor	35
Exercícios	36
Vetores com números aleatórios	36
Exercícios	38
Aula 7: Matrizes	38
Declaração de matriz	38
Inicialização de matriz	40
Exercícios	42
Matriz com valores aleatórios	42
Matrizes de tamanhos variáveis	43
Aula 8: Strings em Java	44

Declaração de string	44
Método length()	44
Método charAt(int)	45
Métodos para conversão: maiúsculos e/ou minúsculos	45
Comparação entre strings	46
Exercícios do Módulo 3	48
Módulo 4: Pesquisa e ordenação	49
Aula 9: Métodos de ordenação	49
Método da bolha (bubble sort)	50
Método inserção direta (insertion sort)	52
Exercício	53
Comparação entre os métodos simples de ordenação	53
Aula 10: Métodos de pesquisa	53
Pesquisa sequencial	54
Pesquisa sequencial ordenada	54
Pesquisa binária	55
Exercício	57
Exercício do Módulo 4	57
Considerações finais	59
Respostas Comentadas dos Exercícios	60
Aula 1	60
Aula 2	60
Aula 3	61
Exercícios do Módulo 1	61
Aula 4	63
Aula 5	64
Exercícios do Módulo 2	66
Aula 6	70
Aula 7	71
Exercícios do Módulo 3	72
Aula 9	79
Aula 10	80
Exercício do Módulo 4	80
Referências Bibliográficas	82

Apresentação

A palavra lógica vem do grego *logos*, que pode ser traduzida como razão, discurso ou linguagem. Os primeiros registros encontrados de lógica na história são de alguns séculos antes de Cristo, nas obras *Organon* e *Metafísica*, de Aristóteles, razão pela qual ele é chamado “pai da lógica”.



No início do século XX, o matemático inglês George Boole consolidou a lógica moderna, trabalhando regras de inferência para analisar tautologias. Essa lógica de predicados ficou conhecida como linguagem booleana.

A lógica passou a ser alvo de estudo na computação a partir da década de 1960. Nesses termos, é a arte de pensar corretamente. Ela estuda a correção do raciocínio. A lógica de programação estuda as técnicas para desenvolver algoritmos computacionais. Um algoritmo é uma sequência de passos que visa atingir um determinado objetivo. Em outras

Manuscritos de Aristóteles.

<http://fabiopestanaramos.blogspot.com.br/2011/10/introducao-logica-aristotelica.html>

palavras, considerando um determinado problema, o algoritmo descreve os passos lógicos para que ele seja resolvido e seu objetivo, atingido.

Um dos grandes desafios dos profissionais da área de TI é estarem aptos a identificar problemas, propor soluções inéditas ou melhorar as que já existem e, ainda, desenvolver projetos que envolvam as mais variadas tecnologias inerentes desta área. Saber usar os recursos tecnológicos e computacionais na implementação e manutenção de novos produtos, serviços e/ou processos é essencial para sua sobrevivência em um mercado extremamente competitivo. A lógica de programação vem ao encontro dessas necessidades, pois seu principal objetivo é estudar e implementar soluções para os mais variados problemas encontrados em nossa sociedade. E é isso o que o espera nesta disciplina, caro aluno, obter os fundamentos da lógica de programação para ajudá-lo a construir soluções em Java.

Módulo 1: Introdução ao Java

Caro aluno, este módulo apresenta a linguagem de programação Java focando o estudo de lógica de programação. Com Java, poderemos desenvolver e testar vários algoritmos, que serão analisados e discutidos. Para isso, iremos enfatizar todos os detalhes da linguagem essenciais para tais ações. Vamos lá.

Aula 1: Primeira aplicação em Java

Conceitos preliminares – ambiente de programação

A linguagem Java é orientada a objetos em sua essência e é utilizada no desenvolvimento de diversos tipos de aplicação. Dentre elas, podemos citar: projetos com banco de dados (local e distribuído), projetos para aplicações em baixa e alta plataforma e aplicações para a *web* – comércio eletrônico, aplicações para *smartphones*, jogos, entretenimento, dentre outras. Algumas de suas principais características são: descende das linguagens C / C++, é compilada e interpretada, portátil, suporta concorrência e programação distribuída.

Um programa Java consiste em partes chamadas *classes*, nas quais os atributos (dados) e os métodos (funções) estão declarados e implementados. Os métodos realizam determinadas tarefas e retornam informações em seu término. Os programas também fazem uso de bibliotecas de classes já existentes, conhecidas como *Java APIs* (*Application Programming Interfaces*).

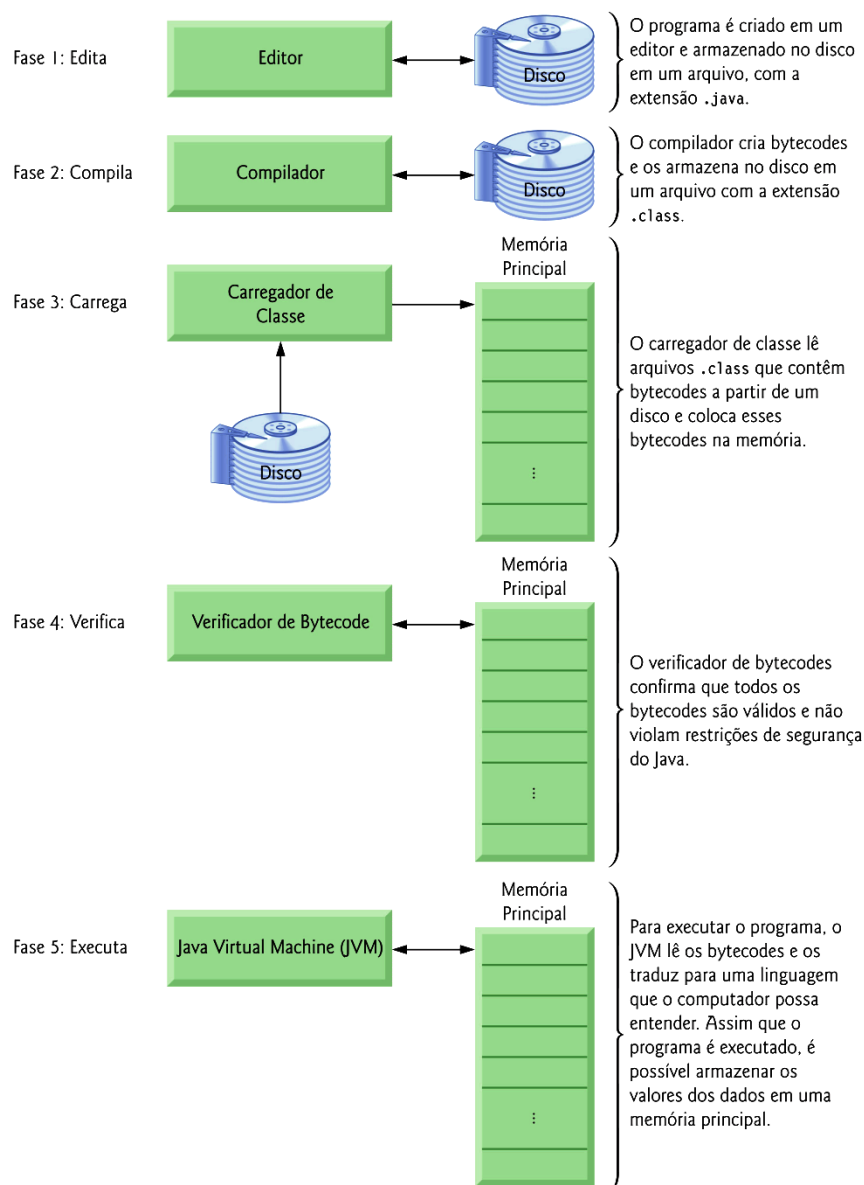
Programas Java normalmente passam por cinco fases:

- 1) Edição → O programador escreve o programa e armazena-o em disco.
- 2) Compilação → O compilador cria *bytecodes* a partir do programa.
- 3) Carga → O carregador de classe armazena *bytecodes* na memória.
- 4) Verificação → O verificador de *bytecodes* confirma que eles não violam restrições de segurança.
- 5) Execução → A JVM (*Java Virtual Machine*) produz *bytecodes* em linguagem de máquina.



Fonte: <http://pplware.sapo.pt/informacao/java-com-mega-patch-de-correcao-de-42-vulnerabilidades/>

A figura 1 ilustra um ambiente de desenvolvimento Java típico.

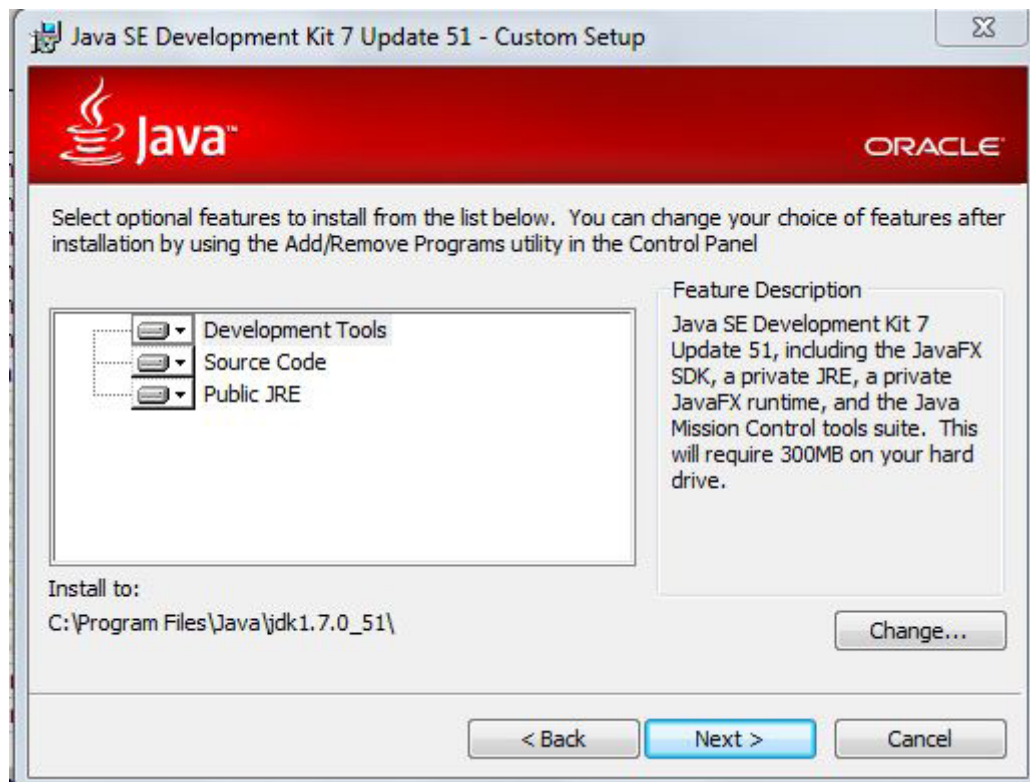


Ambiente de desenvolvimento Java típico. Fonte: Deitel e Deitel, 2005, p. 9.

Para criar aplicativos Java, é necessário ter instalado em sua máquina o pacote de desenvolvimento java JDK (*Java Development Kit*), seguindo os passos:

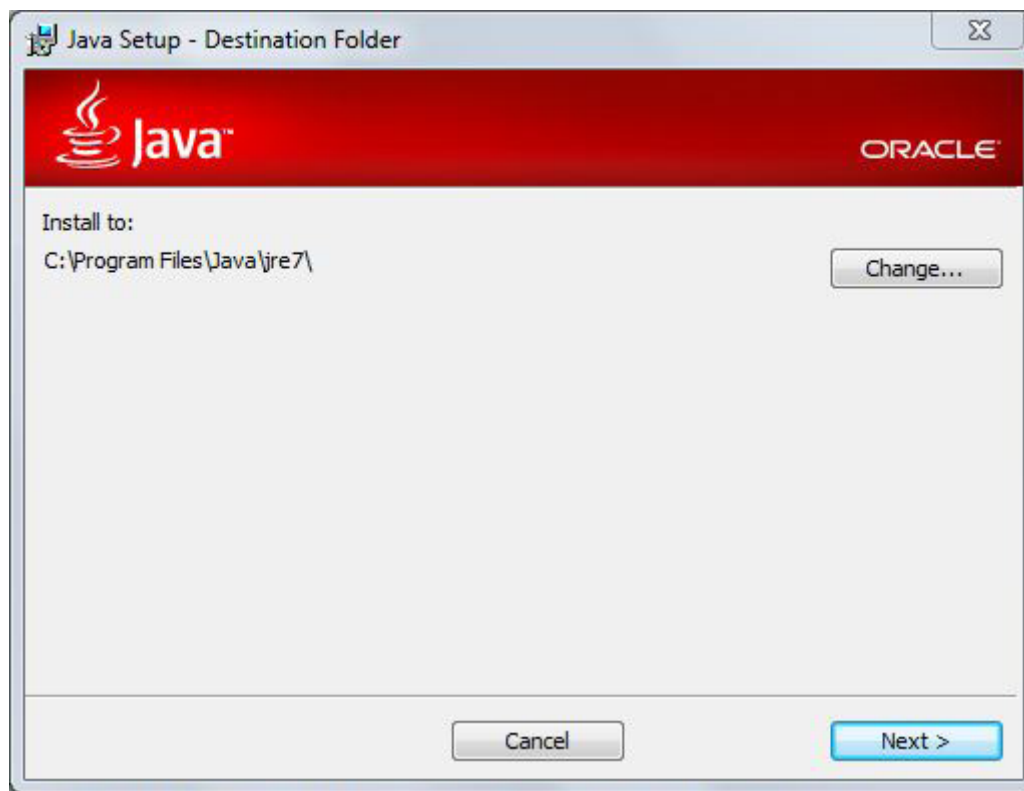
- 1) Acesse o link: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Obs: caso esse link não funcione mais, localize a versão mais recente do JDK e siga os passos aqui descritos.
- 2) Na página, escolha a opção *download JDK* (arquivo: `jdk-7u51-windows-x64.exe` ou `jdk-7u51-windows-i586.exe`) de acordo com a arquitetura Windows que possui. Como todo aplicativo Windows, a instalação é simples. Obs: caso use outra plataforma que não a Windows, baixe o *kit* adequado.

As figuras 2 e 3 ilustram o passo a passo dessa instalação.



Instalação do JDK – passos iniciais. Fonte: instalação do *kit* JDK.

Nesse instante, aguarde para que todos os arquivos que compõem o *kit* JDK sejam instalados em seu computador.



Instalação do JDK – Passo finais. Fonte: instalação do *kit* JDK.

Uma vez instalado o *kit* JDK em seu computador, ele está preparado para executar aplicações em Java. Para executar aplicações em linha de comando, é necessário configurar as variáveis de ambiente. Assim, para que isso ocorra, faça:

- 1) Painel de Controle → Sistema → Configurações Avançadas do Sistema → Variáveis de Ambiente. Na janela Variáveis de Sistema, localize a linha onde está definido o *path*.
- 2) Selecione a opção editar e, em seguida, adicione o caminho da pasta bin (por exemplo, C:\Program Files\Java\jdk1.7.0_51\bin) da instalação Java à variável PATH. Se já existir uma variável PATH, use o separador ; para separar os caminhos.

Pronto. Agora, com o ambiente de programação instalado em sua máquina, podemos começar com os primeiros passos em Java.

Primeiros passos

Quando nos deparamos com estudantes e até mesmo profissionais que estão aprendendo Java, percebe-se que muitas vezes não sabem que existem formas para compilar e executar suas aplicações usando linhas de comando. Em alguns casos, o programador está tão acostumado a usar alguma IDE (*integrated development environment*) que acaba por não saber como realizar o trabalho “manual” para testar suas aplicações. Existem certificações, por exemplo, a OCJP 6, em que o candidato precisa dominar os comandos `javac` e `java`. Uma IDE para desenvolvimento Java será apresentada em outra disciplina, pois o objetivo desta é apresentar as estruturas básicas da linguagem com foco em lógica de programação.

Ao apresentar uma linguagem de programação, é muito comum partir de um programa exemplo simples, a fim de examiná-lo cuidadosamente para que os alunos possam se ambientar e dar os primeiros passos no uso da linguagem. Seguindo esse raciocínio, vamos a um exemplo clássico, o famoso “OlaMundo”. Veja o código no quadro 1.

Quadro 1: Código-fonte do programa OlaMundo

```
1 // Programa Ola Mundo
2 public class OlaMundo
3     public static void main(String[] args) {
4         System.out.println("Ola mundo!");
5     }
6 }
```

Nota: Os números de linha não fazem parte do programa; eles foram adicionados para referência.



Para digitar esse programa, recomendo o uso de um editor simples, como o Notepad, o Bloco de Notas ou o Notepad++ (este é o melhor). Após editar o código do programa, você deve salvá-lo. O nome do arquivo a ser gravado deve ser exatamente igual ao nome que aparece após a palavra *class* na primeira linha do programa e deve ter *.java* como sufixo. Em nosso exemplo, esse nome ficará: `OlaMundo.java`. Preste atenção

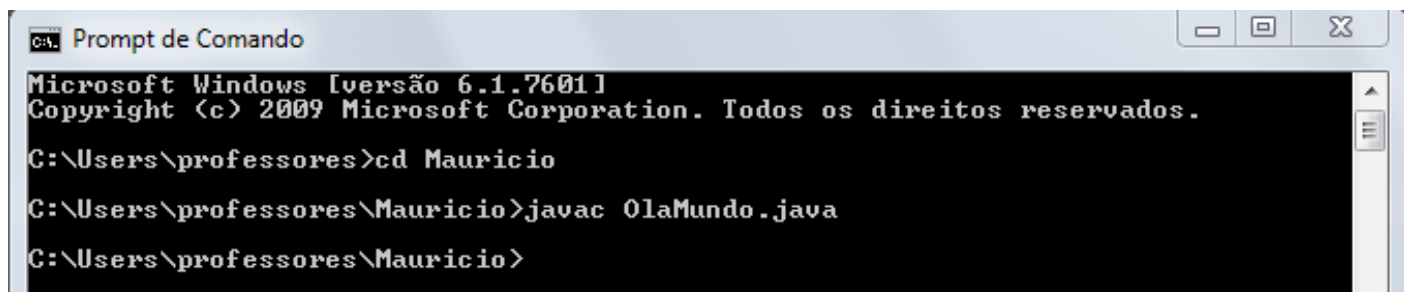
ao digitar o nome, respeitando os caracteres maiúsculos e minúsculos, pois a linguagem Java é *case sensitive* (sensível ao tipo de caixa das letras).

Após a gravação, considerando que ele foi gravado na pasta `C:\Users\professores\Mauricio`, é necessário compilar o arquivo-fonte. Para isso, abra um *prompt* de comando: *Iniciar* → *Acessórios* → *Prompt de Comando* (usando o ambiente Windows). Use o comando `CD` para mudar para a pasta onde o arquivo `OlaMundo.java` foi gravado.

Para compilar o programa, utilize o compilador *javac*, como em:

`C:\Users\professores\Mauricio> javac OlaMundo.java`

A figura 4 ilustra esse processo.



```

C:\Users\professores>cd Mauricio
C:\Users\professores\Mauricio>javac OlaMundo.java
C:\Users\professores\Mauricio>
  
```

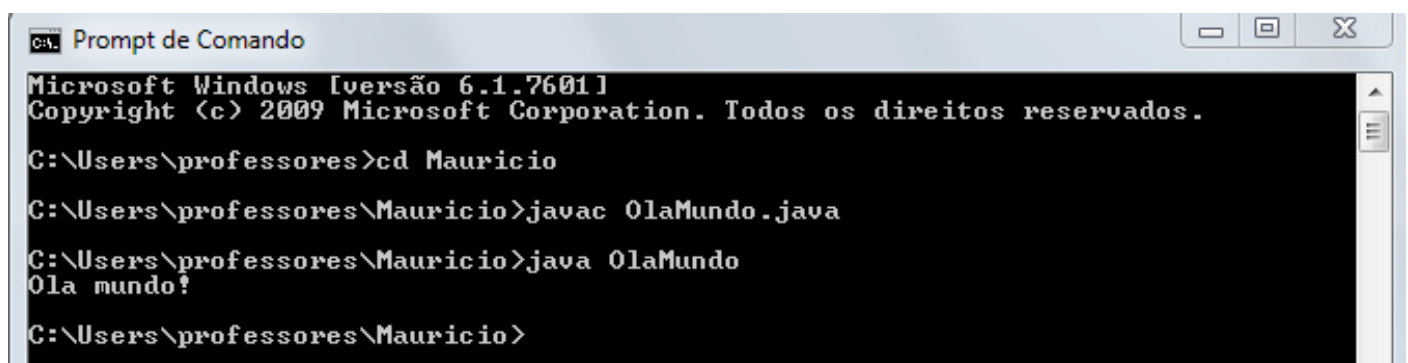
Prompt de comando para compilar.

Caso não haja erros sintáticos (de compilação), um arquivo binário com sufixo `.class` será criado. Em nosso exemplo, teríamos o arquivo: `OlaMundo.class`. Caso haja erros de compilação, será necessário editar novamente o programa-fonte a fim de corrigi-los.

Uma vez compilado o arquivo-fonte, é gerado o arquivo binário. Então é o momento de executar a aplicação. Para isso, na linha de comando, usaremos o comando *java*. Veja:

`C:\Users\professores\Mauricio> java OlaMundo`

A figura 5 ilustra esse processo.



```

C:\Users\professores>cd Mauricio
C:\Users\professores\Mauricio>javac OlaMundo.java
C:\Users\professores\Mauricio>java OlaMundo
Ola mundo!
C:\Users\professores\Mauricio>
  
```

Prompt de comando para executar.

Repare que, nessa linha de comando, não é necessário colocar o sufixo `.class`.

Ao executar esse programa, deverá ser exibido na tela:

Ola mundo!

Se isso não ocorrer, retorne ao editor e verifique se cada linha digitada está igual à da listagem apresentada no quadro 1. Repita os passos para compilar com `javac` e executar com `java`.

Entendendo o primeiro programa

O quadro 1 apresenta o arquivo-fonte do programa `OlaMundo.java`. A seguir, será discriminada cada linha desse arquivo para que você possa compreendê-lo bem.

Linha 1: `//Programa Ola Mundo`

Comentários de linha iniciam-se com: `//`

- Comentários são ignorados durante a execução do programa.
- Documentam e descrevem o código.
- Fornecem legibilidade ao código.
- Comentários tradicionais: `/* ... */`

`/* Este é um comentário tradicional. Ele pode se estender por várias linhas */`

Linha 2: `public class OlaMundo {`

Inicia a declaração de classe para a classe `OlaMundo`.

- Cada programa Java tem pelo menos uma classe definida pelo usuário.
- Palavra-chave: palavras reservadas para uso pelo Java.
- Palavra-chave `class` seguida pelo nome da classe (identificador).
- Atribuindo nomes de classes: coloque a inicial de cada palavra em maiúscula, como em: *ExemploDeNomeDeClasse*.
- A chave esquerda `{` inicia o corpo de toda a classe, que termina com a chave direita `}` na linha 6.

Identificador Java

- Série de caracteres consistindo em letras, dígitos, sublinhados (`_`) e sinais de cifrão (`$`).
- Não pode ser iniciado com um dígito, não pode ter nenhum espaço.
- Exemplos válidos: `cWelcome1`, `$value`, `_value`, `button7t`.
- `7button` não é válido, pois se inicia com número.

Observações:

- 1) O Java diferencia letras maiúsculas de minúsculas, assim, `a1` e `A1` são diferentes.
- 2) O nome do arquivo deve ter exatamente o mesmo nome da classe, respeitando letras maiúsculas e minúsculas. Para o nosso exemplo: `OlaMundo.java`.

Linha 3: `public static void main(String[] args) {`

Parte de todo aplicativo Java:

- Aplicativos começam a executar pelo método `main`.
 - Parênteses indicam que `main` é um método.
 - Aplicativos Java contêm um ou mais métodos.
- Métodos podem realizar tarefas e retornar informações:
 - `void` significa que `main` não retorna nenhuma informação.
- A chave esquerda, `{`, inicia o corpo de declaração do método, e a chave direita, `}`, termina o corpo (linha 5 do programa).

O método `main`, por definição, deve ser `public` e `static` e ter como argumento `String [] args` (um vetor de strings).

Por ora, faremos assim e, em tempo oportuno, melhores definições serão apresentadas.

```

Linha 4: System.out.println("Ola mun-
do!");

```

Instrui o computador a realizar uma ação:

- Imprime *strings* de caracteres.
 - *String*: série de caracteres entre aspas duplas.
- Espaços em branco em *strings* não são ignorados pelo compilador.

System.out:

- Objeto de saída-padrão.
- Imprime na janela de comando (isto é, *prompt* do MS-DOS para ambientes Windows).

Método *System.out.println*:

- Exibe uma linha de texto.
- Instruções terminam com ponto-e-vírgula (;).

Exercício

Abra o terminal (linha de comando) e crie uma pasta com seu nome. Armazene todas as aplicações que criarmos nela para facilitar seu trabalho.

Crie um novo programa em Java, colocando as seguintes informações pessoais na tela:

- Nome Endereço Telefone celular E-mail

Aula 2: Tipos de dados, variáveis e operadores

Tipos de dados primitivos e declaração de variáveis

A linguagem Java possui vários tipos de dados primitivos. A tabela 1 apresenta uma descrição resumida de cada um e sua capacidade de armazenamento.

Tabela 1 – Tipos de Dados em Java

Tipo	Domínio	Valor inicial	Tamanho
byte	-128 até 127	0	8 bits
short	-32.768 até 32.767	0	16 bits
int	-2.147.483.648 até 2.147.483.647	0	32 bits
long	-9223372036854775808 até 9223372036854775807	0L	64 bits
float	+/- 3.40282347E+38F (aproximadamente 7 dígitos significativos)	0.0f	32 bits
double	+/- 1.79769313486231570E+308 (15 dígitos significativos)	0.0d	64 bits
char	0 até 65536	'\u0000' (Null)	16 bits
boolean	true / false	false	1 bit

Fonte: Deitel e Deitel, 2010.

As variáveis são usadas para armazenar e manipular dados na memória RAM do computador. Na linguagem Java, elas devem ser declaradas para que possam ser usadas. A declaração de uma variável consiste em

definir um nome (identificador) associado a um tipo de dados. O acesso a uma variável é realizado pelo seu identificador.

Declaração de variáveis:
<tipo de dado> <lista de identificadores separados por vírgulas> ;

Lembre-se de que um identificador deve ser iniciado com uma letra, underline (_), ou sinal de dólar (\$), seguidos de letras e/ou dígitos e que existe uma diferenciação entre letras maiúsculas e minúsculas.

O quadro 2 ilustra alguns exemplos de declarações de variáveis em Java.

Quadro 2: Exemplos de Declarações em Java

```
1  int numero, idade;
2  double preco, total;
3  char resp;
```

Os identificadores devem ser criados segundo a regra apresentada, porém não podemos usar nomes já definidos na linguagem Java. Java possui várias palavras reservadas, ilustradas na tabela 2.

Tabela 2: Palavras reservadas do Java

abstract	do	implements	private	throw
boolean	double	import	protected	throws
break	else	instanceof	public	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	super	volatile
class	float	new	switch	
continue	for	null	synchronized	
default	if	package	this	

Fonte: Deitel e Deitel, 2010.

Como visto na tabela 1, as variáveis, quando declaradas, possuem valores iniciais, porém, caso você queira, novos valores iniciais podem ser definidos ao declará-las. O quadro 3 ilustra alguns exemplos.

Quadro 3: Inicialização de variáveis

```
1  int contador = 1;
2  double total = 0;
3  char resp = true;
```

A declaração de uma variável pode ser feita em qualquer parte do programa, porém recomenda-se que isso ocorra sempre no início de um bloco. Em um mesmo bloco, uma variável deve ser única, ou seja, não se pode declarar duas ou mais variáveis com o mesmo identificador. As variáveis sempre terão escopo local ao bloco onde foram declaradas. Caso haja

blocos e sub-blocos, aquelas declaradas no bloco podem ser usadas nos sub-blocos a ele pertencentes, porém as declaradas no sub-bloco não podem ser usadas em seu "bloco-pai".

O programa exemplo do quadro 4 ilustra duas situações de declarações envolvendo blocos e sub-blocos: uma válida e outra inválida.

Quadro 4: Exemplo do uso de declarações em blocos

```

1 // Exemplo de declarações
2 public class Declaracoes
3     public static void main(String[] args) {
4         int numero = 5;
5         System.out.println(numero); // Válido
6         // sub-bloco do bloco do metodo main
7         {
8             final double pi = 3.1415; // final faz pi ser constante
9             System.out.println(numero + " " + pi); // Válidos!
10        }
11        System.out.println(pi); // Inválidos: pi não é visível
12    }
13 }

```

Operadores

A linguagem Java possui vários operadores. Estes são utilizados para manipular valores armazenados nas variáveis. Podemos agrupá-los da seguinte forma:

- Operador de atribuição.
- Operadores aritméticos.
- Operadores relacionais.
- Operadores lógicos.
- **Operador de atribuição e atribuição composta**

O operador de atribuição é responsável por alterar o conteúdo armazenado em uma variável. Na linguagem Java, usa-se o símbolo `=`. O quadro 5 ilustra alguns exemplos.

Quadro 5: Exemplos de atribuições

```

1 // Exemplo de atribuições
2 int x,y;
3 double a;
4 x = 10;
5 y = x;
6 a = 5.5

```

Alguns cálculos que usam variáveis para acumular resultados em operações aritméticas podem ser realizados usando os operadores de atribuição composta, como ilustrado na tabela 3.

Tabela 3: Operadores de atribuição composta

Expressão	Significado
<code>x += y</code>	<code>x = x + y</code>
<code>x -= y</code>	<code>x = x - y</code>
<code>x *= y</code>	<code>x = x * y</code>
<code>x /= y</code>	<code>x = x / y</code>

Operadores aritméticos

Os operadores aritméticos básicos utilizados em Java são ilustrados na tabela 4.

Tabela 4: Operadores aritméticos

Operação	Símbolo	Exemplo
Adição	+	<code>A + B</code>
Subtração	-	<code>A - B</code>
Multiplicação	*	<code>A * B</code>
Divisão	/	<code>A / B</code>
Resto da divisão entre inteiros	%	<code>A % B</code>

O quadro 6 ilustra um exemplo com os operadores aritméticos.

Quadro 6: Exemplo do uso de operadores aritméticos

```
1 // Exemplo de operações aritméticas
2 public class Aritmeticas {
3     public static void main(String[] args) {
4         int dez = 5+5;
5         int vinte = 30 - 10;
6         int trinta = 15 * 2;
7         int quarenta = 80 / 2;
8         int tres = 11 % 8;
9         System.out.println(dez + " " + vinte + " " + trinta);
10        System.out.println(quarenta + " " + tres);
11    }
12 }
```

Observações quanto aos resultados que envolvem operandos de diferentes tipos básicos:

Ao usar operações aritméticas em um programa, o resultado será do mesmo tipo que os operandos envolvidos. Assim, se usarmos operandos inteiros (char, int, byte, long), teremos como resultado valores inteiros. Da mesma forma, se usarmos operandos reais (float ou double), o resultado será real. Se, em uma expressão aritmética, houver operandos inteiros e reais combinados, o resultado será real.



Um detalhe que deve ser enfatizado é o fato de que a ordem de precedência dos operadores aritméticos é o mesmo expresso pela matemática. Para dar prioridade a uma operação, utilizamos apenas parênteses. Exemplos:

$$(A + B) * C$$
$$(A - B) * ((C + D) * E)$$

Dentre os operadores aritméticos, existem os de incremento e de decremento, que merecem nossa atenção.

- **Operador de incremento e de decremento**

Os operadores de incremento e decremento somam 1 ou subtraem 1 da variável a ele associada. Por exemplo, se a variável A tiver valor 5 e realizarmos A++, então A ficará com 6. O mesmo vale para o decremento. Esses operadores poderão ser pré ou pós (incremento/decremento). Nesse caso, existem diferenças. O quadro 7 ilustra um exemplo do uso deles.

Quadro 7: Exemplo do uso de operadores de incremento e decremento

```

1 // Exemplo de operadores ++ e --
2 public class Incrementa_Decrementa {
3     public static void main(String[] args) {
4         int a = 5, b, c, d;
5         a++; // a ficou com 6
6         // primeiro copia a para b e depois incrementa a
7         b = a++; // b ficou com 6 e a com 7
8         // primeiro decrementa b e depois copia seu valor para c
9         c = --b; // c ficou com 5 e b também ficou com 5
10        // primeiro copia c para d e depois decrementa c
11        d = c--; // d ficou com 5 e c com 4
12        System.out.println(a + " " + b + " " + c + " " + d);
13    }
14 }

```

• Operadores relacionais

Os operadores relacionais são usados para auxiliar o controle do fluxo de um programa. Eles são os responsáveis pelas operações de comparações entre duas variáveis ou entre variáveis e constantes. Assim, eles sempre retornam um valor *boolean* (*true* ou *false*). A tabela 5 mostra todos os operadores relacionais da linguagem Java.

Tabela 5: Operadores relacionais

Operador	Significado	Exemplo
==	Igual	x == 3
!=	Diferente (não igual)	x != 3
<	Menor que	x < 3
>	Maior que	x > 3
<=	Menor ou igual	x <= 3
>=	Maior ou igual	x >= 3

• Operadores lógicos

Assim como os operadores relacionais, os operadores lógicos também auxiliam o controle do fluxo do programa, porém eles usados para combinar duas ou mais condições em um único teste. Eles também retornam valor *boolean* (*true* ou *false*). A tabela 6

ilustra os operadores lógicos utilizados na linguagem Java.

Tabela 6: Operadores lógicos

Operador	Significado
&&	Operação lógica E (AND)
	Operação lógica OU (OR)
!	Negação lógica

Para exemplificar o funcionamento de uma operação lógica, vamos considerar duas condições, A e B (que podem assumir F ou V). A tabela 7 ilustra os resultados.

Tabela 7: Tabelas-verdade dos operadores lógicos

A	B	A && B	A B	!A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Exercícios

1) Digite o seguinte código, compile-o, execute-o e analise as respectivas saídas de dados.

```
public class Exercicio_aula_1_b
{
    public static void main(String[] args) {
        int x=5, y=10, z;
        float a=4.5f, b;
        double c, d;
        char e;
        long f;
        boolean teste;
        x++; //incrementa x em uma unidade
        System.out.println("X = "+x);
        z = y--; // copia para z o valor de y e depois decrementa y
        System.out.println("Y = "+y+" Z = "+z);
        b = a+x; //operações entre int e float resultam em float
        System.out.println("A = "+a+" B = "+b);
        c = 15/2; // mesmo c sendo double, como 15 e 2 são int, resulta na parte inteira
        d = a; // double é também um tipo real assim como float, atribuição apenas converte
        System.out.println("C = "+c+" D = "+d);
        e = 'a';
        System.out.println("E = "+e);
        f = x+y+z; // f é long, inteiro e a soma x+y+z resulta em int, apenas converte.
        System.out.println("F = "+f);
        teste = z>y; // teste recebe true se o teste for verdadeiro, caso contrario recebe
false
        System.out.println("Teste = "+teste);
    }
}
```

- 2) Elabore um programa em Java que crie as variáveis nota1, nota2 e nota3 do tipo double. Em seguida, atribua as seguintes notas: 5.0, 3.5 e 9.5, respectivamente. Calcule a média aritmética entre elas e armazene em outra variável double declarada como media. Mostre a média calculada.
- 3) Desenvolva um programa em Java que calcule e exiba a o preço de venda de um produto qualquer cujo preço de custo é R\$ 37,00 com uma porcentagem para o vendedor de 12% e impostos de 26,95%, sabendo que o preço de venda é dado pelo preço de custo, ao qual são adicionados o lucro do vendedor e os impostos (ambos aplicados ao preço de custo). Crie as variáveis que julgar necessárias e mostre o resultado na tela.

Aula 3: Entrada e saída de dados

Entrada de dados em Java

É muito comum associarmos conhecimentos que temos em uma linguagem para aprender outra. Assim, programadores que usam a linguagem C estão acostumados com a função `scanf()` para realizar a entrada de dados via teclado e esperam encontrar algo parecido em Java. Porém ler dados em Java é uma tarefa

um pouco mais elaborada quando comparada à linguagem C. Não vamos entrar em detalhes sobre o paradigma orientado a objetos nesta disciplina, mas uma das formas mais comumente usadas para realizar a entrada de dados é por meio da classe *Scanner*, pertencente ao pacote *java.util*. A seguir, vamos discriminar os passos necessários para a realização de entradas de dados em Java.

- 1) A primeira tarefa a fazer é importar o pacote *java.util* no início do programa. Assim:
import java.util.Scanner;
- 2) Na sequência, deve-se criar um objeto da classe *Scanner*. Esse objeto será usado para realizar todas as entradas de dados desejadas. Assim:
Scanner entrada = **new** Scanner(System.in);

Explicação: a palavra reservada *new* é usada para instanciar um objeto, isto é, alocar memória para o objeto e, ainda, criar uma referência para ele. Nesse exemplo, o objeto criado é *entrada*. Por meio dele, criado e instanciado, é possível realizar as entradas de dados de acordo com os tipos de variáveis desejados.

Feito isso, para ler um dado pelo teclado, veja o que deve ser feito para cada um dos diferentes tipos.

- Dados do tipo inteiro (int):
int num;
System.out.println("Digite um valor inteiro:");
num = entrada.**nextInt()**;
- Dados do tipo real (float ou double):
float preco;
double taxa;
System.out.println("Digite o preço do produto:");
preco = entrada.**nextFloat()**;
System.out.println("Digite a taxa de juros:");
taxa = entrada.**nextDouble()**;
- Dados do tipo *string* (simples, sem espaços em branco):
String palavra;
System.out.println("Digite uma palavra:");

palavra = entrada.**next()**;

- Dados do tipo *string* (composto, ou seja, contendo espaços em branco):

String frase;

System.out.println("Digite uma frase:");

frase = entrada.**nextLine()**;

- Em alguns casos, pode ser necessário ler valores numéricos e, na sequência, ler *strings*. Quando isso ocorre, o *buffer* (memória do teclado) deve ser esvaziado a fim de evitar entradas de dados indesejadas. Caso isso ocorra, devemos fazer:

String nome;

int idade;

System.out.println("Digite sua idade:");

idade = entrada.**nextInt()**;

entrada.**nextLine()**; // esvaziando o buffer do teclado

System.out.println("Digite o seu nome completo:");

nome = entrada.**nextLine()**;

- Dados do tipo caracter único (char)

char letra;

System.out.println("Digite uma letra:");

letra = entrada.**nextLine().charAt(0)**;

Observação:

A instrução *nextLine()* faz a leitura de uma string, como visto anteriormente. Ao adicionarmos *charAt(0)*, realizamos a recuperação do primeiro caracter dela. Os caracteres existentes em uma string são todos arranjados sequencialmente, e o primeiro está na posição 0 (zero), por isso *charAt(0)*, pois o 0 indica qual desejamos recuperar. Os demais caracteres serão ignorados.

Saída de dados em Java

A linguagem Java possui vários métodos para realizar a saída de dados. Os mais comuns são os que já temos usado em nossos exemplos, *print()* ou *println()*, da classe *System*. A diferença entre eles é que

o método `println()`, após ecoar a mensagem, faz com que o cursor avance para a próxima linha, o que não ocorre com o `print()`.

Para enviar uma mensagem para a tela, basta fazer:

```
System.out.println ("Aqui vem sua mensagem");
```

Caso seja necessário, juntamente com a mensagem, ecoar o valor de alguma variável, basta usar o operador "+", que efetuará a concatenação da mensagem com o valor. Nesse caso, o valor é convertido para *string*, pois esses métodos realizam sempre saídas nessa forma. Veja:

```
String nome;
```

```
System.out.println("Digite o seu nome completo:");
```

```
nome = entrada.nextLine();
```

```
System.out.println("Nome digitado: "+ nome);
```

Caracteres especiais, como "\n", que avançam para a próxima linha, podem ser inseridos em qualquer parte da mensagem a ser mostrada. Além disso, não há limites para as concatenações desejadas. Recomendo aqui usar bom senso e não criar mensagens enormes que acabem por dificultar a leitura de seu código-fonte. Veja:

```
String nome;
```

```
int idade;
```

```
System.out.println("Digite sua idade:");
```

```
idade = entrada.nextInt();  
entrada.nextLine(); // esvaziando o buffer do teclado
```

```
System.out.println("Digite o seu nome completo:");
```

```
nome = entrada.nextLine();  
System.out.println ("Seu nome é: "+nome +" \nVocê tem:"+idade+ "anos!");
```

Agora que já aprendemos a realizar entradas e saídas de dados, vamos a um exemplo completo. O quadro 8 ilustra um programa que faz a entrada das medidas de um terreno retangular (largura e comprimento) e, na sequência, calcula e mostra a área e o perímetro dele.

Quadro 8: Exemplo do uso de entrada e saída de dados (arquivo: Terreno.java)

```

1 // Exemplo de Entrada e Saída
2 import java.util.Scanner;
3 public class Terreno {
4     public static void main(String[] args) {
5         float larg, comp, area, per;
6         Scanner ler = new Scanner(System.in);
7         System.out.println("Digite a largura do terreno:");
8         larg = ler.nextFloat();
9         System.out.println("Digite o comprimento do terreno:");
10        comp = ler.nextFloat();
11        area = larg*comp;
12        per = 2*(larg+comp);
13        System.out.println("Area = "+area+"\nPerimetro = "+per);
14    }
}

```

Exercícios

- 1) Digite o exemplo exposto no quadro 8. Compile-o e execute-o.
- 2) Elabore um programa que receba pelo teclado dois valores inteiros e os armazene nas variáveis A e B. Em seguida, troque os conteúdos das variáveis de tal modo que, em A, esteja o valor de B e, em B, o valor de A. Veja o exemplo: caso sejam lidos para A o valor 5 e para B o valor 10, seu programa deverá fazer com que A receba 10 e B, 5. Mostre os valores de A e B antes e depois da troca.

Exercícios do Módulo 1

- 1) Assinale os identificadores corretos:

- a) teste
- b) x1
- c) 2abc
- d) Cod_Produto
- e) Nome Cliente
- f) Porcentagem%
- g) Nota1
- h) Media Final

- 2) Analise o código-fonte abaixo e escreva quais são os valores das variáveis A, B e C mostrados na tela.

```

public class Exercicio2 {
    public static void main(String[] args) {
        int A=10, B, C=0;

        B = A++;
        C += ++B;
        System.out.println("A = "+A+"B = "+B+"C = "+C);
    }
}

```

- 3) Elabore um programa em Java chamado Desconto, que, tendo como dado de entrada um valor real que indica o preço de um produto, calcule e mostre o novo preço, sabendo que ele sofreu um desconto de 10%.

- 4) Elabore um programa em Java - Conversor de Reais para Dólares – que, tendo como dados de entrada o valor em reais e a cotação do dólar, calcule e mostre o valor em dólares correspondente.
- 5) Elabore um programa em Java – Número de Três Algarismos – que, tendo como dado de entrada um valor inteiro de três algarismos (vamos considerar que o usuário digitará um valor nessa forma), mostre os algarismos em separado. Por exemplo, supondo o número lido igual a 384, o programa deverá mostrar: Centena = 3, Dezena = 8 e Unidade = 4.
- 6) Elabore um programa em Java – Conversor de Tempo – que, tendo como dado de entrada um tempo em segundos, calcule e mostre em horas, minutos e segundos. Veja o exemplo: considerando que o tempo de entrada seja 8.750 segundos, o programa mostrará que esse tempo equivale a: 2 horas, 25 minutos e 50 segundos.

Módulo 2: Estruturas de controle

Caro aluno, este módulo 2 apresenta as estruturas fundamentais para o controle do fluxo de um programa em Java. Aqui são abordadas as estruturas de seleção simples, composta e múltipla, as estruturas de repetições e muitas aplicações que as utilizam.

Aula 4: Estruturas de seleção ou decisão

Uma estrutura de seleção deve ser usada em um programa quando há necessidade de escolher entre uma determinada instrução ou outra. Para que essa escolha ocorra, alguma ação e/ou condição deve ser analisada e/ou testada. Essa condição é uma expressão relacional e/ou lógica e sempre retornará *true* ou *false*.

Estrutura de seleção simples if

Uma estrutura de seleção simples é realizada em Java com a instrução *if*. Veja a sintaxe:

```
if (expressão)
{
    <instrução 1>;
    <instrução 2>;
    ...
}
```

Quando o processador encontra o bloco de instruções acima, inicialmente a <expressão> é avaliada. Caso o resultado dessa avaliação seja um valor *true*, o conjunto de instruções dentro do bloco delimitado por { e } será executado. Caso resulte em um valor *false*, todo o bloco de instruções será ignorado, e o fluxo de execução do programa seguirá para a próxima instrução abaixo do bloco *if* { }. O bloco de instruções deverá estar entre { e } sempre que nele existirem duas ou mais instruções que devam ser executadas caso a <expressão> seja avaliada como verdadeira. Se existir apenas uma instrução nesse bloco, então { e } podem ser omitidos.

O quadro 9 ilustra um programa em que dois valores inteiros são dados como entrada: a velocidade máxima permitida em uma rodovia e a velocidade com que um veículo nela trafega. O programa deverá mostrar na tela a mensagem "Motorista, você foi multado! Está XX acima da velocidade permitida YY", caso sua velocidade seja superior à permitida.

Quadro 9: Exemplo do uso de estrutura de seleção simples (arquivo: Velocidade.java)

```
1 //Exemplo de Estrutura de Seleção Simples
2 import java.util.Scanner;
3 public class Velocidade {
4     public static void main(String[] args) {
5         int vel, vel_max, dif;
```

```

6      Scanner ler = new Scanner(System.in);
7      System.out.println("Digite a velocidade máxima permitida na rodovia:");
8      vel_max = ler.nextInt();
9      System.out.println("Digite a velocidade que o veículo trafega:");
10     vel = ler.nextInt();
11     dif = vel - vel_max;
12     if (dif > 0 )
13     {
14         System.out.println("Motorista você foi multado!");
15         System.out.println("Está "+dif+" acima da vel. permitida "+vel_
max);
16     }
17 }
18 }

```

No exemplo apresentado, ao executar o comando *if* na linha 13, a expressão (*dif > 0*) é avaliada. Caso ela resulte em *true* (verdadeira), o bloco de instruções compreendido por { e } entre as linhas 14 e 17 será executado. Caso seja *false*, o bloco { e } entre as linhas 14 e 17 será ignorado.

Estrutura de seleção composta *if ... else*

Uma estrutura de seleção composta é realizada em Java com a instrução *if... else*. Veja sintaxe:

```

if (expressão)
{
    <Bloco true>;
}
else
{
    <Bloco false>;
}

```

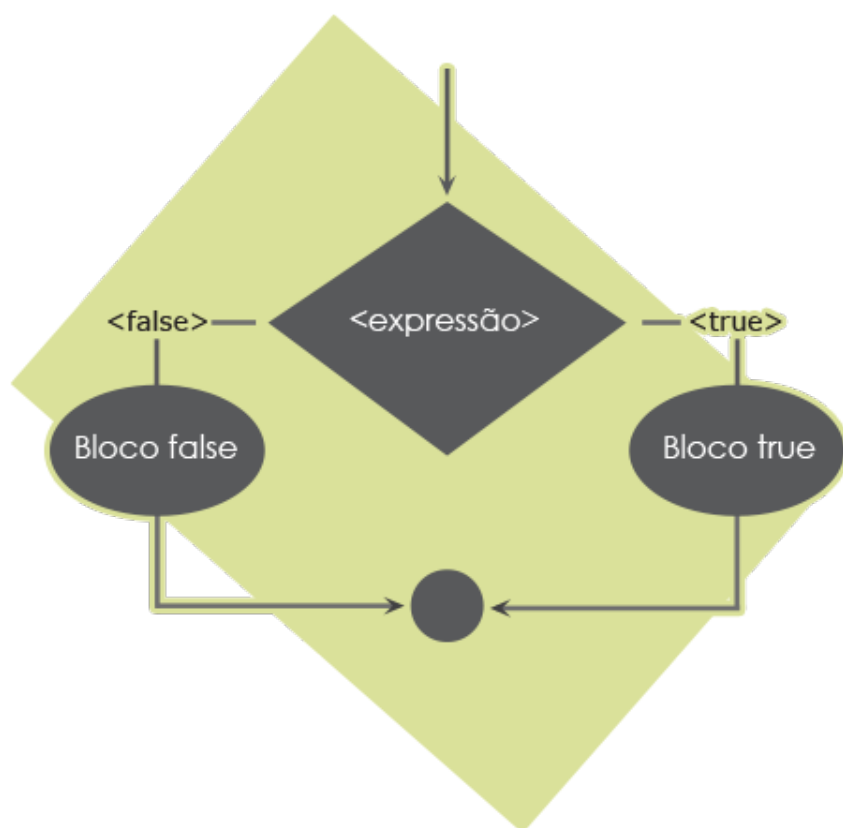
Quando o processador encontra o bloco de instruções acima, inicialmente, a <expressão> é avaliada. Caso o resultado seja *true*, o conjunto de instruções dentro do { <Bloco true> } será executado, e o { <Bloco false> }, ignorado. Caso a <expressão> resulte em um valor *false*, o { <Bloco true> } será ignorado, e o { <Bloco false> }, executado. Nessa estrutura de seleção, sempre um bloco é executado: ou o <Bloco true>, ou o <Bloco false>. Assim como na estrutura de seleção simples, se existir apenas uma instrução nos Blocos true ou false, então os delimitadores de início e fim { e } podem ser omitidos.

Duas observações muito importantes a ser feitas:

Pode-se ter uma instrução *if* sem o bloco *else*, mas o contrário não é verdadeiro. Toda instrução *else* deve estar associada a uma instrução *if*. Cada instrução *if* pode ter no máximo uma instrução *else*.

Ao escrever a instrução *else*, não se deve colocar nenhuma expressão relacional e/ou lógica, pois a sua execução sempre ocorrerá quando a expressão da instrução *if* for *false* (somente nessa condição).

É muito comum, ao consultar livros que abordam lógica de programação, encontrarmos diagramas que ilustram o raciocínio empregado. Em programação, esses diagramas são chamados de fluxogramas, e os símbolos utilizados expressam a ação que o programa deve realizar. A figura 6 ilustra um trecho de fluxograma que expressa a lógica empregada na estrutura de seleção composta.



Fluxograma da estrutura seleção composta: Fonte: criação do autor.

O quadro 10 ilustra um programa que faz uso da estrutura de seleção composta. Nele, um valor inteiro é lido pelo teclado, e o programa verificará se esse valor é par ou ímpar. A lógica aqui apresentada analisa o resto da divisão do número lido por 2 e, se esse resto for 0 (zero), conclui-se que este valor é par, caso contrário, o valor lido é ímpar.

Quadro 10: Exemplo do uso da estrutura de seleção composta (arquivo: ParImpar.java)

```
1 //Exemplo de Estrutura de Seleção Composta
2 import java.util.Scanner;
3 public class ParImpar {
4     public static void main(String[] args) {
5         int num;
6         Scanner ler = new Scanner(System.in);
7         System.out.println("Digite um valor inteiro:");
8         num = ler.nextInt();
9         if (num % 2 == 0 )
10             System.out.println(" O valor "+num+" é par!");
11         else
12             System.out.println(" O valor "+num+" é ímpar!");
13     }
14 }
```

Observem que, na linha 11 do quadro 10, na qual aparece a instrução *if*, o operador de igualdade utilizado é "==" pois o operador "=" é para realizarmos atribuições e não comparações. Nesse programa, uma das frases ("O valor xx é par!" ou "O valor xx é ímpar!") será ecoada na tela mediante o teste (`num%2 == 0`), da linha 11, ser *true* ou *false*. Note ainda que, nos "blocos" *true* e *false*, não foram utilizados os delimitadores { e }, isso porque apenas uma instrução os compõe, sendo estes desnecessários.

Exercícios

- 1) Elabore um programa em Java que leia três valores inteiros (A, B e C) e armazene o maior deles na variável maior. Mostre na tela o valor da variável maior.
- 2) Elabore um programa em Java que leia um ano (valor inteiro) e verifique se é ou não bissexto. Um ano para ser bissexto deve: a) ser múltiplo de 400 ou b) ser múltiplo de 4 e não de 100.

A instrução *switch* analisa a <expressão> ou <variável> e, baseada nesse valor (resultado da expressão ou do valor armazenado na variável), desvia o controle do fluxo do programa para o comando *case*, que contém em sua definição o <valor x> igual ao valor da expressão ou variável resultante. Todos os blocos definidos em seus respectivos comandos *case* devem ser encerrados com o comando *break*;, pois este força o fluxo do programa a saltar para a instrução seguinte ao *switch*. Caso o comando *break*; seja omitido, o programa continuará a executar os demais *cases* até encontrar o fechamento do *switch*.

O quadro 11 ilustra um exemplo de programa que faz uso da estrutura *switch... case*. O programa obtém pelo teclado a leitura de um caracter (*char*) e, caso ele seja uma vogal, mostra na tela a frase correspondente.

Estrutura de seleção múltipla *switch... case*

A instrução *switch* é classificada como estrutura de seleção, pois escolhe um entre vários valores possíveis, ou seja, realiza uma seleção dentre múltiplas possibilidades. Diferentemente da estrutura de seleção composta, na qual apenas um entre dois blocos poderiam ser escolhidos, a estrutura *switch* escolhe um entre vários blocos definidos. Veja como:

```
switch (<expressão> ou <variável>)
{
  case <valor1>: <Bloco 1 a ser executado>; break;
  case <valor2>: <Bloco 2 a ser executado>; break;
  case <valorn>: <Bloco n a ser executado>; break;
  default: <Bloco de programa a ser executado caso
           nenhuma das condições anteriores forem executadas>;
           break;
}
```

Quadro 11: Exemplo do uso da estrutura switch...case (arquivo: ExemploSwitch.java)

```

1  //Exemplo de Estrutura switch ... case
2  import java.util.Scanner;
3  public class ExemploSwitch {
4      public static void main(String[] args) {
5          char letra;
6          Scanner ler = new Scanner(System.in);
7          System.out.println("Digite uma letra:");
8          letra = ler.nextLine().charAt(0);
9          switch(letra)
10         {
11             case 'a':
12                 case 'A': System.out.println ("Vogal A"); break;
13             case 'e':
14                 case 'E': System.out.println ("Vogal E"); break;
15             case 'i':
16                 case 'I': System.out.println ("Vogal I"); break;
17             case 'o':
18                 case 'O': System.out.println ("Vogal O"); break;
19             case 'u':
20                 case 'U': System.out.println ("Vogal U"); break;
21             default: System.out.println("Você não digitou uma vogal!"); break;
22         }
23     }
24 }

```

No exemplo do quadro 11, o comando *default* – linha 21 - dentro do *switch* será executado apenas quando nenhum dos *cases* anteriormente declarados for verdadeiro.

Importante notar que, quando mais de um valor de *case* puder resultar numa mesma ação, basta colocá-los em sequência. Observe o caso das letras maiúsculas e minúsculas, que são igualmente vogais no exemplo do quadro 11.

Aula 5: Estruturas de repetição

Uma estrutura de repetição deve ser usada em um programa quando há necessidade de fazer que determinado bloco de instrução seja executado várias vezes. O bloco permanecerá em execução até que determinada condição se torne falsa. Essa condição é uma expressão relacional e/ou lógica e sempre retornará *true* ou *false*.

Nesta aula, são apresentadas três formas distintas de realizar a repetição. Não se pode dizer que uma é melhor que a outra, apenas que seu comportamento se adapta a determinados trechos do programa de

forma mais clara, adequada e organizada. Cabe ao programador escolher de qual das formas de repetição fará uso em seus programas.

Repetição com teste no início

Como o próprio nome diz, o teste, ou melhor, a expressão relacional e/ou lógica, será avaliado logo no início da estrutura de repetição e, caso seja verdadeiro, o bloco pertencente a ele será executado até que essa condição se torne falsa. Em Java, essa estrutura é realizada pelo comando *while*, cuja sintaxe é mostrada abaixo.

```
while (<expressão>)
{
    <Bloco de instruções a ser executado enquanto a
    <expressão> for true>
}
```

A semântica do *while* funciona assim: a <expressão> é avaliada e, caso ela seja *true*, o bloco de instruções compreendido entre { e } será executado.

Ao terminar a execução, o controle automaticamente volta para a <expressão> definida no comando, e tudo se repete, ou seja, ela será novamente avaliada e, enquanto for verdadeira, o bloco de instruções será novamente executado. O laço de repetição finalizará quando a <expressão> retornar um valor *false*. Importante notar que, se a condição não retornar um valor *false*, o *while* nunca finaliza.

A regra para a delimitação do bloco de instruções permanece a mesma, assim, caso haja apenas uma instrução, os delimitadores { e } podem ser omitidos. Eles são obrigatórios se o bloco contiver duas ou mais instruções.

O quadro 12 ilustra um exemplo que faz uso da instrução *while*. Neste programa, os números inteiros compreendidos entre 1 e 100 serão somados um a um. O programa termina quando o número atinge a contagem de 101 (primeiro valor inteiro maior que 100) e, assim, o valor da soma é mostrado na tela do computador.

Quadro 12: Exemplo do uso do comando while (arquivo: ExemploWhile.java)

```
1 //Exemplo de Estrutura de Repetição - comando while
2 public class ExemploWhile {
3     public static void main(String[] args) {
4         int num=1, soma=0;
5         while (num <= 100) {
6             soma += num;
7             num++;
8         }
9         System.out.println("Soma dos números entre 1 e 100 = "+soma);
10    }
11 }
```

No programa exemplo do quadro 12, na linha 8, encontra-se a instrução *soma += num;* cujo significado é: soma-se à variável *soma* o valor de *num*, acumulando-se o cálculo na variável *soma*. Como a instrução da linha 9 incrementa a variável *num* em uma unidade, garante-se que todos os valores, começando por 1, serão somados e acumulados na variável *soma*.

Exercícios

- 1) Elabore um programa em Java para somar os 10 primeiros valores inteiros lidos pelo teclado. Mostrar a soma na tela. Veja o exemplo: para os seguintes números lidos: 5, 12, 7, 22, 45, 4, 8, 85, 3 e 16, o programa calculará e mostrará a soma: $5 + 12 + 7 + 22 + 45 + 4 + 8 + 85 + 3 + 16$.
- 2) Elabore um programa em Java que, considerando uma classe com 20 alunos, calcule e mostre o total de homens e de mulheres da turma. Para isso, leia o sexo de cada aluno (basta uma letra 'm' para masculino ou 'f' para feminino).
- 3) Substitua a instrução *while* do programa do quadro 12 pelo trecho abaixo e teste o programa.

```
while (true) {
    soma += num;
    ++num;
    if (num > 100)
        break;
}
```

Por que esse trecho funciona? O que significa *while (true)*?

O exemplo ilustrado no quadro 13 mostra o uso da estrutura *do... while()*. Neste exemplo, é solicitado ao usuário que digite uma nota válida (entre 0 e 10); caso ela não esteja nesse intervalo, a solicitação será repetida. Veja:

Quadro 13: Exemplo do uso do comando *do... while* (arquivo: ExemploDoWhile.java)

```
1 //Exemplo de Estrutura de Repetição - comando do ... while
2 import java.util.Scanner;
3 public class ExemploDoWhile {
4     public static void main(String[] args) {
5         double nota;
6         Scanner ler = new Scanner(System.in);
7         do{
8             System.out.println("Digite uma nota: ");
9             nota = ler.nextDouble();
10            if ((nota < 0) || (nota > 10)) {
```

O que a instrução *break*; causa ao comando *while (true)*?

Repetição com teste no final

A diferença dessa estrutura de repetição daquela com teste no início é que, primeiramente, o bloco a ser repetido é executado, para depois a expressão relacional e/ou lógica ser avaliada. Caso ela seja *true*, o bloco será novamente executado. Essa repetição também terminará quando a expressão avaliada se tornar *false*. O comando em Java que realiza essa repetição é o *do ... while ()*. Veja a sintaxe a seguir:

```
do
{
    <Bloco de instruções a ser executado>
} while (<expressão>);
```

A repetição com teste no início não garante que o bloco de instruções pertencente à estrutura seja executado ao menos uma vez. Isso porque, caso a expressão seja falsa logo no primeiro teste, a repetição é abortada. Já na estrutura em que a repetição é realizada no final do bloco, garante-se que o bloco seja executado pelo menos uma vez.


```

11         System.out.println("Nota invalida! Redigite-a...");
12     }
13     }while ((nota<0)|| (nota>10));
14     System.out.println("Voce digitou a nota: "+nota);
15 }
16 }

```

Exercício

Elabore um programa em Java para calcular a média das idades de um grupo de pessoas. A repetição do `while()` deverá ser encerrada quando uma idade inválida (≤ 0) for lida pelo teclado. Mostre a média entre as idades lidas.

Repetição com variável de controle

Essa estrutura de repetição é realizada em Java com o comando *for*. Esse comando é muito útil e versátil, pois é possível criar repetições de diversas formas diferentes, desde simples, com contadores, até complexas, com diversas variáveis de controle. A sintaxe é a seguinte:

```

for ( <inicializações> ; <expressão> ; <operações> )
{
    <Bloco de instruções a ser executado>
}

```

Como apresentado, a estrutura *for* possui três regiões:

- 1) <Inicialização>:** É reservada para inicializar uma ou mais variáveis. Cada inicialização deve ser separada por vírgulas. São aceitas declarações de variáveis na inicialização, bem como outras instruções válidas do Java. Essa região poderá ficar vazia, porém o ponto e vírgula que a finaliza é obrigatório. O que estiver contido nessa região é executado apenas uma vez.
- 2) <Expressão>:** É a condição (relacional e/ou lógica) para a continuação do *for*. Se a expressão for avaliada como verdadeira, o bloco de instruções será executado. Caso seja omitida, ela sempre assumirá o valor *true*, e portanto, será uma repetição infinita. Quando essa expressão for avaliada como *false*, a repetição termina.
- 3) <Operação>:** Depois de executar o bloco de instruções, o controle é direcionado para essa região, na qual uma ou várias operações ali definidas serão executadas. Caso seja necessário executar mais do que uma operação, estas devem estar separadas por vírgulas. Essa região também poderá ser omitida.

O quadro 14 ilustra o uso de um comando *for* em Java. Neste programa, os números compreendidos entre 1 e 100 serão mostrados na tela do computador.

Quadro 14: Exemplo do uso do comando for (arquivo: ExemploFor_1.java)

```

1 //Exemplo de Estrutura de Repetição - comando for
2 public class ExemploFor_1 {
3     public static void main(String[] args) {
4         int num;
5         for (num = 1; num <=100; num++) {
6             System.out.println(num);
7         }
8     }
9 }

```

No exemplo apresentado, na linha 6, temos um comando *for* básico. Nele, a variável *num* é inicializada com valor 1 e, em seguida a expressão (*num <=100*) é avaliada. Caso ela seja *true*, o comando da linha 7 (impressão do valor em *num*) será executado. Ao terminar essa impressão, o controle é direcionado para a operação (*num++*), no qual a variável *num* é incrementada em uma unidade. Após essa operação, o controle volta a avaliar a <expressão> e assim, todo o processo é repetido enquanto ela retornar *true*.

Você pode observar toda a versatilidade do comando *for* pela sequência a seguir, que mostra como o trecho entre as linhas 5 e 8 do programa do quadro 14 pode ser escrito de diferentes maneiras:

- 1) **for** (int num = 1; num <=100; num++) {
 System.out.println(num);
}
- 2) **int** num=1;
 for (; num <=100; num++) {
 System.out.println(num);
 }
- 3) **int** num=1;
 for (; **num** <=100;) {
 System.out.println(num);
 num++;
 }
- 4) **int** num=1;
 for (System.out.println("Começou o laço for") ; num <=100;) {
 System.out.println(num);
 num++;
 }
- 5) **int** num=1;
 for (; ;) {
 System.out.println(num);
 num++;
 if (num > 100)
 break;
 }

O exemplo do quadro 15 mostra o uso de um comando *for* mais elaborado. Neste programa, são inicializadas duas variáveis e também são usadas duas operações sobre elas. Note que, para tais ações, faz-se uso de vírgulas para separá-las.

Quadro 15: Exemplo do uso do comando for (arquivo: ExemploFor_2.java)

```

1 //Exemplo de Estrutura de Repetição - comando for
2
3 public class ExemploFor_2 {
4     public static void main(String[] args) {
5         int a,b;
6
7         for (a=1, b=2; a+b<=20 ; a++, b+=2 ) {
8             System.out.println(a+ " +b+ " +(a+b));
9         }
10    }
11 }

```

No exemplo, no comando *for* da linha 7, as variáveis *a* e *b* são inicializadas com os valores 1 e 2, respectivamente. A expressão $(a+b \leq 20)$ é avaliada e, caso seja verdadeira, o comando de impressão da linha 8 será executado. Após essa impressão, as operações *a++* e *b+=2* serão executadas nessa ordem, e o controle voltará a avaliar a expressão. O comando *for* terminará quando a expressão retornar *false*. Note que tanto as variáveis inicializadas quanto as operações realizadas estão separadas por vírgulas. Esse programa mostrará na tela:

```

1 2 3
2 4 6
3 6 9
4 8 12
5 10 15
6 12 18

```

Observações:

- Um comando *for* (; ;) provoca uma repetição infinita em seu bloco de instruções, pois não há variáveis inicializadas, não há expressão de término e também não há operações a ser realizadas. Nesse caso, uma instrução *break*; pode ser usada para finalizá-lo.
- Um comando *for* (; cont<=10;) é equivalente a *while* (cont<=100).
- Importante notar que, mesmo não usando inicializações, expressões e/ou operações, os dois ponto e vírgula são obrigatórios.

Exercícios

- 1) Elabore um programa em Java que, usando a estrutura de repetição *for*, leia 10 valores inteiros pelo teclado, calcule e mostre a soma desses valores.
- 2) Modifique o programa do quadro 14 substituindo o trecho entre as linhas 5 e 8 pelas cinco diferentes possibilidades apresentadas. Teste cada uma delas.

Comandos break e continue

Como já o instigamos a verificar em exemplos anteriores, o comando *break* provoca a interrupção de um laço de repetição. O *break* é usado em estruturas de repetição *while*, *do... while* e *for* e também junto ao comando *switch... case*. Quando usado em uma repetição, ele provoca uma interrupção imediata do laço, transferindo o controle para a instrução seguinte. O quadro 16 ilustra um exemplo do uso do comando *break*.

Quadro 16: Exemplo do uso do comando break (arquivo: ExemploBreak.java)

```

1      //Exemplo do uso do comando break
2      public class ExemploBreak {
3          public static void main(String[] args) {
4              int a=1;
5              while (true) {
6                  System.out.println(a);
7                  a++;
8                  if (a==10) break;
9              }
10         }
11     }

```

No exemplo ilustrado, temos na linha 6 um laço de repetição infinito. Nele, o valor da variável *a* é impresso na tela e em seguida incrementado. Quando a variável atingir o valor 10, o comando *break* da linha 9 será executado, a repetição infinita será abortada, e o programa terminará.

O comando *continue* é usado somente em estruturas de repetição. Quando é executado, o controle volta imediatamente ao início da repetição para nova avaliação da <expressão> que define o término dela; caso a expressão ainda seja true, o bloco volta a ser executado. Normalmente, usa-se o comando *continue* associado a um teste *if*. O quadro 17 ilustra um exemplo.

Quadro 17: Exemplo do uso do comando continue (arquivo: ExemploContinue.java)

```

1      //Exemplo do uso do comando continue
2      public class ExemploContinue {
3          public static void main(String[] args) {
4              int a=0;
5              while (a<=10) {
6                  a++;
7                  a++;
8                  if (a==5) continue;
9                  System.out.println(a);
10             }
11         }
12     }

```

O programa apresentado no quadro irá imprimir o valor da variável *a*, porém o valor 5 será omitido, pois, quando essa variável possuir esse valor, o comando *continue*, da linha 8, será executado, e o controle voltará para o início da repetição, na linha 6.

Exercício

Elabore um programa Java que imprima os números divisíveis por 3 entre 0 e 300, usando o comando sugerido abaixo:

```
for ( int i=0; i<=300; i++) {
    if (i%3>0)
        continue;
    //aqui você deve imprimir o valor do múltiplo de 3
    na tela
}
```

Exercícios do Módulo 2

- Elabore um programa em Java que leia um valor inteiro e mostre todos os seus divisores. Por exemplo, para o valor 20, os divisores serão: 1, 2, 4, 5, 10 e 20.
- Elabore um programa em Java que leia um valor inteiro e verifique se ele é um número primo ou não. Um número, para ser primo, deve possuir apenas dois divisores: 1 e ele próprio.
- Elabore um programa em Java que leia dois valores reais (operandos) e um caracter (operador aritmético: +, -, * ou /) e realize a operação desejada com os operandos, de modo a encerrar o programa quando o operador aritmético lido for '#'.
- Elabore um programa em Java que calcule e mostre o valor da seguinte soma:

$$S = 1 + 2 + 3 + 4 + 5 + \dots + 100$$
- Elabore um programa que calcule o IMC (Índice de Massa Corporal) de cada pessoa em um grupo de 10 pessoas. Mostre a situação de cada uma. O IMC é calculado pela fórmula:

$$IMC = \frac{\text{Peso}}{\text{Altura}^2}$$

IMC	Situação
< 18	Abaixo do peso
>=18 e < 25	Peso normal
>= 25 e < 30	Acima do peso
>= 30	Obesidade
- Elabore um programa em Java que leia três valores inteiros (a, b, e c) e, em seguida, coloque-os em ordem crescente, de modo que o menor esteja em a, o intermediário, em b e o maior, em c. Repita o programa sempre que o usuário desejar.
- O número 3.025 tem a seguinte característica:

$$30 + 25 = 55$$

$$55^2 = 3.025$$

Elabore um programa em Java para mostrar todos os números de quatro algarismos que possuem essa mesma característica.
- Elabore um programa em Java que calcule e mostre o valor da seguinte soma:

$$S = 1/1 + 2/2 + 4/3 + 8/4 + 16/5 + \dots + 2^{20}/20$$

Módulo 3: Estruturas de dados básicas

O módulo 3 apresenta as estruturas de dados básicas do Java, que correspondem a variáveis compostas, capazes de armazenar mais do que um valor. Aqui, são abordadas as estruturas de dados básicas, vetores e matrizes, e também falaremos sobre *strings*. Os vetores e as matrizes são chamados de variáveis compostas homogêneas, pois todos os valores neles armazenados são do mesmo tipo.

Aula 6: Vetores (*arrays*)

Um vetor (*array*) é uma estrutura de dados que ocupa uma porção de memória alocada contigualmente, capaz de armazenar vários valores. Esses valores são comumente chamados de elementos do vetor e podem ser usados individualmente tanto para leitura como para escrita. Cada valor está associado a um número que identifica sua posição de armazenamento, conhecido como índice. A quantidade de elementos a ser alocada em um vetor é um valor fixo, e todos os elementos que serão nele armazenados são do mesmo tipo, por isso o nome variável composta homogênea ou estrutura de dados homogênea.

Declaração de vetores

A linguagem Java, assim como as linguagens C e C++, trabalha com vetores *zero-based*, isto é, os índices dos vetores iniciam-se com o valor zero. Portanto, um vetor com 10 elementos possui índices que vão de 0 a 9. Os vetores podem ser declarados como segue:

```
<tipo de dado> <identificador_do_vetor> [ ];
```

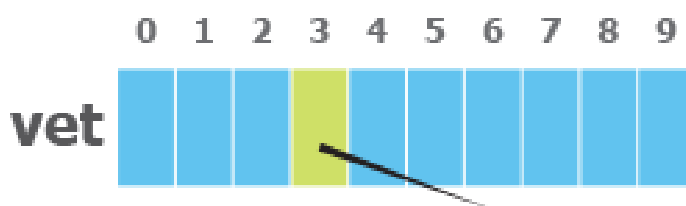
Um vetor que é apenas declarado ainda não está pronto para ser usado. Para que possa armazenar valores, a memória necessita ser alocada com a respectiva quantidade de elementos desejada. Para alocar memória em Java, usaremos o operador *new*. O qua-

dro 18 mostra duas formas diferentes para declarar e alocar vetores com 10 valores inteiros.

Quadro 18: Exemplo de declaração de vetor do tipo *int*

```
1  int vet [ ];
2  vet = new int [10];
3  // ou simplesmente...
4  int vet [ ] = new int[10];
```

O vetor *vet*, declarado no exemplo, pode ser esquematizado da seguinte forma:



O tamanho do vetor poderá ser uma variável do tipo *int*, desde que ela contenha algum valor previamente atribuído ou lido. O quadro 19 ilustra o exemplo de declaração de um vetor capaz de armazenar 100 notas de alunos.

Quadro 19: Exemplo de declaração de vetor do tipo double

```
1  int n = 100; //total de alunos
2  double notas= new double [n]; //vetor notas com 100 valores para double
```

Inicialização de vetores

Os vetores podem ser inicializados no momento de sua declaração. Se isso ocorrer, não há necessidade de alocação de memória com o operador *new*, uma vez que a quantidade de valores inicializados determina o tamanho do vetor (quantidade de elementos). O quadro 20 ilustra dois exemplos de inicializações.

Quadro 20: Exemplo de declaração e inicialização de vetores

```
1  int a [ ] = {1,2,3,4,5};
2  // nesta declaração, a[0] == 1, a[1]==2, a[3]==2 ...
3  String nomes [ ] = {"João", "Maria", "José" };
4  // nesta declaração, nomes[0] tem o valor "João", nomes[1] o valor "Maria" ...
```

Manipulação dos elementos de um vetor

Os elementos de um vetor devem ser manipulados um a um, tanto na leitura quanto na escrita. Assim, o ideal para alimentar um vetor com valores fornecidos pelo usuário é efetuar a leitura desses elementos dentro de uma estrutura de repetição. O quadro 21 ilustra um exemplo de entrada dos elementos de um vetor pelo teclado e, em seguida, a impressão deles.

Quadro 21: Exemplo do uso de vetores em Java (arquivo: ExemploVetor.Java)

```
1  import java.util.Scanner;
2  public class ExemploVetor {
3      public static void main(String[] args) {
4          int a [ ] = new int[5]; // declarando um vetor a com 5 elementos
5          int i; // variável indexadora do vetor
6          Scanner ler = new Scanner(System.in);
7          for (i=0; i<5; i++) {
8              System.out.print( "A ["+i+"] = ");
9              a[i] = ler.nextInt(); // lendo cada elementos em sua devida posição
10         }
11         System.out.println("Vetor lido:");
12         for (i=0; i<5; i++){
13             System.out.println("A ["+i+"] = "+a[i]);
14         }
15     }
16 }
```


Exercícios

- 1) Melhore o exemplo do quadro 20 para que, na impressão do vetor lido, sejam indicados o menor e o maior valor do vetor. Exemplo: considerando como lido o vetor

	0	1	2	3	4	5	6	7	8	9
a	10	28	3	15	12	6	45	9	11	25

A impressão deverá ser:

A[0] = 10

A[1] = 28

A[2] = 3 → "Menor Valor"

A[4] = 15

A[5] = 12

A[6] = 45 → "Maior Valor"

A[7] = 9

A[8] = 11

A[9] = 25

- 2) Elabore um programa em Java que crie um vetor capaz de armazenar 10 notas de alunos. Após a leitura delas, calcule e mostre a média aritmética entre as 10 notas do vetor (somar todas e em seguida dividir a soma por 10).

Vetores com números aleatórios

Um número é chamado de aleatório quando não segue nenhum padrão para ser gerado. Em muitas aplicações, como em cálculos estatísticos, é necessário trabalhar com vetores grandes e aleatórios. Um vetor é dito aleatório quando seus elementos são gerados automaticamente pelo sistema, não seguindo nenhum padrão. A linguagem Java possui uma classe chamada *java.util.Random* que contém alguns métodos estáticos (*static*), ou seja, não necessitam ser instanciados, o que pode nos auxiliar a gerar e organizar tais vetores. Para usá-la, basta criar um objeto. Veja:

Random gerador = **new Random();**

O objeto *gerador* será usado para gerar os valores aleatórios. Caso sejam valores inteiros, basta fazer *gerador.nextInt();* se forem valores booleanos, pode-se fazer *gerador.nextBoolean();* e assim por diante. O quadro 22 ilustra um exemplo gerar números inteiros entre 0 e 99.

Quadro 22: Exemplo do uso de vetor aleatório em Java (arquivo: ExemploRandom.Java)

```

1  import java.util.Random;
2  public class ExemploRandom {
3      public static void main(String[] args) {
4          int a [] = new int[5] , i;
5          Random gerador = new Random();
6          System.out.println("Gerando elementos aleatorios no vetor:");
7          for (i=0; i<5; i++){
8              a[i] = gerador.nextInt(100);
9          }
10         System.out.println("Vetor Gerado:");
11         for (i=0; i<5; i++){
12             System.out.println("A["+i+"]="+a[i]);
13         }
14     }
15 }
```

No programa ilustrado, um vetor com cinco valores é gerado aleatoriamente, e o parâmetro 100 do método `nextInt()` é usado para definirmos a faixa de valores a ser gerados, dessa forma, entre 0 e 99.

Caso necessite gerar valores, por exemplo, entre 10 e 19, basta fazer o deslocamento desejado. O quadro 23 ilustra um exemplo no qual um vetor de valores do tipo `int` é gerado aleatoriamente com valores entre 10 e 19.

Quadro 23: Exemplo do uso de vetor `int` aleatório em Java (arquivo: `ExemploRandom2.java`)

```

1  import java.util.Random;
2  public class ExemploRandom2 {
3      public static void main(String[] args) {
4          int a [] = new int[5], i;
5          Random gerador = new Random();
6          System.out.println("Gerando elementos aleatorios no vetor:");
7          for (i=0; i<5; i++){
8              a[i] = gerador.nextInt(10)+10;
9          }
10         System.out.println("Vetor Gerado:");
11         for (i=0; i < a.length ; i++){
12             System.out.println("A["+i+"]="+a[i]);
13         }
14     }
15 }

```

Observações:

Como um array é um objeto, existem alguns métodos que podem ser usados em sua manipulação. Um deles é o método `length`, que retorna a quantidade de elementos que um array possui.



Sintaxe: `<nome_array>.length`

O quadro 23, ao imprimir o vetor gerado aleatoriamente, faz uso do método `length` (equivalente ao valor 5, nesse caso).

Para gerar números aleatórios reais, podemos usar o método `nextDouble()` associado ao nosso objeto *gerador* da classe *Random*. Porém os números pertencerão ao intervalo $[0, 1[$, ou seja entre 0 e 1, não incluindo o valor 1. Para fazer com que os valores gerados fiquem entre 0 e 99, deve-se multiplicar por 100. O quadro 24 ilustra tal exemplo.

Quadro 24: Exemplo do uso de vetor double aleatório em Java (arquivo: ExemploRandom3.Java)

```

1  import java.util.Random;
2  public class ExemploRandom3 {
3      public static void main(String[] args) {
4          double a [] = new double[5], i;
5          Random gerador = new Random();
6          System.out.println("Gerando elementos aleatorios no vetor:");
7          for (i=0; i<5; i++){
8              a[i] = gerador.nextDouble()*100;
9          }
10         System.out.println("Vetor Gerado:");
11         for (i=0; i < a.length ; i++){
12             System.out.println("A["+i+"]="+a[i]);
13         }
14     }
15 }

```

Exercícios

- 1) Melhore o exemplo do quadro 21 para que, na impressão do vetor lido, sejam indicados o menor e o maior valor.
- 2) Elabore um programa em Java que crie um vetor capaz de armazenar 10 notas de alunos entre 0.0 e 10.0. Gere as notas dos alunos de forma aleatória, calcule e mostre a média aritmética entre elas.

Aula 7: Matrizes

Uma matriz é uma estrutura de dados homogênea bidimensional. Pode-se dizer que uma matriz é um vetor em que cada elemento é outro vetor. Assim como um vetor, ela deve ser declarada e instanciada (alocar memória para seus elementos).

Declaração de matriz

A forma para se declarar uma matriz em Java é ilustrada a seguir:

```
<tipo de dado> <identificador_do_vetor> [ ];
```

O quadro 25 ilustra um exemplo de declaração de uma matriz capaz de armazenar cinco linhas, cada uma contendo quatro colunas, ou seja, uma matrix 5 x 4.

Quadro 25: Exemplo de declaração de uma matriz do tipo int

```
1  int mat [ ][ ] = new int[5][4];
2  // o primeiro [ ] define o número de linhas que a matriz conterá, e o segundo [ ] define o
3  // número de colunas. Assim, nesta declaração, mat é uma matriz de 5 linhas por
4  // 4 colunas.
```

Ao criarmos uma matriz, devemos ter em mente a estrutura de dados criada. Assim, para a declaração acima, a matriz `mat[5][4]` seria esquematizada da seguinte forma:

	0	1	2	3
0				
1				
2				
3				
4				

`mat[3][2]`

Para manipular matrizes, assim como fizemos em vetores, usaremos variáveis auxiliares para indicar a posição de cada elemento a ser acessado (lido e/ou escrito). Como a matriz possui duas dimensões, usaremos duas variáveis indexadoras: uma para indicar a linha e a outra, a coluna da matriz a ser utilizada. O quadro 26 ilustra um exemplo do uso de matriz.

Quadro 26: Gerando uma matriz 5x4 (arquivo: ExemploMatriz1.java)

```
1  public class ExemploMatriz1 {
2      public static void main(String[] args) {
3          int a [][ ] = new int[5][4]; // alocando matriz com 5 linhas e 4 colunas
4          int i,j;
5          for (i=0; i<5 ; i++)
6              for (j=0; j<4; j++ ) {
7                  a[i][j] = (i+1);
8              }
9          System.out.println("Matriz Gerada:");
10         for (i=0; i<5; i++){
11             for (j=0; j<4; j++)
12                 System.out.print(a[i][j]+" ");
13             System.out.println();
14         }
15     }
16 }
```

A matriz gerada no exemplo do quadro 26, que você pode conferir implementando e testando o programa, terá os seguintes valores:

Matriz gerada:

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
5 5 5 5
```

Inicialização de matriz

Da mesma forma que os vetores, as matrizes também poderão ser inicializadas.

Caso essa inicialização seja feita na declaração da matriz, não há necessidade de usar o operador new, isso porque, com os valores inicializados, a memória é alocada com tamanho e espaço suficientes para eles.

O quadro 27 ilustra um exemplo de inicialização de matriz na declaração.

Observem, no exemplo do quadro 27, que os elementos que inicializarão cada linha da matriz devem estar entre os delimitadores { e } e toda a matriz inicializada também deve conter os delimitadores { e }. Tanto os elementos quanto as linhas inicializadas devem estar separados por vírgulas.

Quadro 27: Inicializando uma matriz 5x4 (arquivo: ExemploMatriz2.java)

```
1 public class ExemploMatriz2 {
2     public static void main(String[] args) {
3         int a [][] = {{5,5,5,5},
4                       {4,4,4,4},
5                       {3,3,3,3},
6                       {2,2,2,2},
7                       {1,1,1,1} };
8         int i,j;
9         System.out.println("Matriz Gerada:");
10        for (i=0; i<5; i++){
11            for (j=0; j<4; j++)
12                System.out.print(a[i][j]+" ");
13            System.out.println();
14        }
15    }
16 }
```

Nesse exemplo, o código foi escrito colocando cada inicialização em uma linha diferente apenas para ilustrar a organização lógica dos elementos na matriz, porém isso não é obrigatório: todos os valores inicializados podem estar em uma mesma linha do código-fonte.

Manipulação dos elementos de uma matriz

Os elementos de uma matriz devem ser manipulados um a um, tanto na leitura quanto na escrita. Assim, o ideal para alimentar uma matriz com valores fornecidos pelo usuário é efetuar a leitura desses elementos

dentro de duas estruturas de repetição, uma para indexar cada linha e outra para indexar cada coluna dentro de uma linha específica.

O quadro 28 ilustra um exemplo de entrada dos elementos de uma matriz quadrada 5x5 pelo teclado e, em seguida, mostra os elementos contidos na diagonal principal. A figura 7 ilustra a diagonal principal da matriz.

	0	1	2	3	4
0					
1					
2					
3					
4					

Quadro 28: Diagonal principal de uma matriz 5x5 (arquivo: ExemploMatriz3.java)

```

1  import java.util.Scanner;
2  public class ExemploMatriz3 {
3      public static void main(String[] args) {
4          int a[][] = new int [5][5];
5          int i, j;
6          Scanner ler = new Scanner(System.in);
7          System.out.println("Digite os elementos da Matriz 5x5:");
8          for (i=0; i<5; i++){
9              for (j=0; j<5; j++){
10                 a[i][j] = ler.nextInt( );
11             }
12             System.out.println("Diagonal principal: ");
13             for (i=0; i<5; i++){
14                 System.out.println( a[i][i]) ; // na diagonal principal linhas e
colunas são iguais
15             }
16         }
17     }

```

Exercícios

- 1) Elabore um programa em Java que leia uma matriz quadrada 5x5 e mostre apenas os elementos que compõem a diagonal secundária. A diagonal secundária é formada pelos elementos: a₀₄, a₁₃, a₂₂, a₃₁ e a₄₀. Veja o esquema na figura 8:

	0	1	2	3	4
0					
1					
2					
3					
4					

- 2) Elabore um programa em Java que leia uma matriz 5x5 e armazene o valor da soma dos elementos de cada linha num vetor. Depois imprima a matriz e o vetor lado a lado.

Matriz com valores aleatórios

Assim como para gerar vetores com valores aleatórios, podemos também usar a classe *java.util.Random* para gerar uma matriz aleatória. O quadro 29 ilustra a geração de uma matriz 5x5 contendo valores aleatórios entre 10 e 19.

Quadro 29: Matriz Aleatória 5x5 (arquivo: ExemploMatriz4.java)

```

1  import java.util.Random;
2  public class ExemploMatriz4 {
3      public static void main(String[] args) {
4          int a[][] = new int[5][5];
5          int i,j;
6          Random gerador = new Random();
7          System.out.println("Gerando elementos aleatorios da matriz...");
8          for (i=0; i<5; i++){
9              for (j=0; j<5; j++)
10                 a[i][j] = gerador.nextInt(10)+10;
11            }
12            System.out.println("Matriz Gerada:");
13            for (i=0; i<5; i++) {
14                for (j=0; j<5; j++)
15                    System.out.print(a[i][j]+" ");
16                System.out.println();
17            }
18        }
19    }

```


Observações:

Como visto em vetores, o método *length* retorna a quantidade de elementos de um *array*. O que aconteceria se o usássemos em uma matriz? Para o exemplo do quadro 29, teríamos:



a.length → total de linhas e
a[i].length → total de colunas da linha *i*.

Matrizes de tamanhos variáveis

Em Java, podemos usar matrizes sem definirmos a quantidade de colunas. Com isso, é possível criar uma matriz na qual cada linha teria um número distinto de colunas. O quadro 30 ilustra uma matriz na qual existem 12 linhas (uma para cada mês do ano), porém as colunas indicam quantos dias cada mês possui e, assim, possuem tamanhos variados.

Quadro 30: Matriz com colunas variadas (arquivo: ExemploMatriz5.java)

```

1      public class ExemploMatriz5 {
2          public static void main(String[] args) {
3              int mes[] = {31,29,31,30,31,30,31,31,30,31,30,31};
4              String nome_mes[] = {"Jan", "Fev", "Mar", "Abr", "Mai", "Jun", "Jul",
5                  "Ago", "Set", "Out", "Nov", "Dez"};
6              int calendario[][] = new int[12][];
7              int i,j;
8              for (i=0; i<12; i++){
9                  calendario[i] = new int[ mes[i]]; // alocando o número de colunas
10                 // relativo ao mes
11                 for (j=0; j< mes[i]; j++)
12                     calendario[i][j] = j+1; // numerando os dias 1, 2, ...
13             }
14             System.out.println("Calendário:");
15             for (i=0; i<12; i++){
16                 System.out.print("Mes "+nome_mes[i]+" = ");
17                 for (j=0; j<mes[i]; j++)
18                     System.out.print(calendario[i][j]+" ");
19                 System.out.println();
20             }
21         }
22     }

```

No exemplo apresentado, foram criados dois vetores:

- O vetor *mes*, que contém 12 elementos; cada um indica o total de dias que os meses possuem, partindo-se de janeiro até dezembro.

- O vetor *nome_mes*, também com 12 elementos, nos quais estão armazenados os nomes dos meses correspondentes.

Para trabalhar com colunas diferentes em cada linha, é necessária a alocação individual por coluna. Isso foi feito na linha 9.

Aula 8: Strings em Java

Assim como em C, não existe um tipo de dado *string* em Java. Porém esta linguagem presenteia-nos com uma classe *String*, na qual encontramos diversos métodos para manipularmos *strings* em nossos programas. As classes em Java têm seus identificadores iniciados com letras maiúsculas. Esta aula apresenta alguns desses métodos e como utilizá-los.

Declaração de *string*

String <identificador_da_string> ;

O quadro 31 mostra um exemplo de declaração e uso de *strings*. Neste exemplo, foram declaradas duas: *msg*, que foi instanciada após a declaração, e *prof*, que, em sua declaração, foi inicializada e, por esta razão, não há necessidade do operador *new*.

Quadro 31: Declaração de *strings*

```

1      public class DeclaracaoString {
2          public static void main(String[] args) {
3              String msg;
4              String prof = "Mauricio Duarte";
5              msg = new String("Estou aprendendo Strings...");
6              System.out.println(msg + "com o Prof. "+prof);
7          }
8      }
```

Na aula 3, foi apresentada a maneira para realizar a leitura de um *string* pelo teclado. Assim, apresentaremos agora os métodos mais comumente usados dentre os mais de 50 que a classe *String* possui.

Método *length()*

Length, em inglês significa comprimento, então, o método, como o próprio nome diz, retorna um valor inteiro que indica o tamanho da *string*. O quadro 32 ilustra um exemplo em que uma *string* é lida pelo teclado e, após ser mostrada na tela, também é mostrada a quantidade de caracteres que ela possui.

Quadro 32: Método length() com *strings* (arquivo: ExemploString1.java)

```

1      import java.util.Scanner;
2      public class ExemploString1 {
3          public static void main(String[] args) {
4              String msg;
5              int tam;
6              Scanner ler = new Scanner(System.in);
7              System.out.println("Digite uma frase");
8              msg = ler.nextLine();
9              tam = msg.length();
10             System.out.println("Frase: "+msg+" possui "+tam+" caracteres.");
11         }
12     }

```

Método charAt(int)

O método *charAt* é usado para referenciar cada caracter existente em uma *string*. O parâmetro *int* indica o caracter a ser retornado. Na linguagem Java, o primeiro caracter de uma *string* está na posição 0. O quadro 33 ilustra um exemplo em que um deles é procurado na *string* e, caso seja encontrado, a posição de sua primeira ocorrência é impressa na tela.

Quadro 33: Método charAt() com *Strings* (arquivo: ExemploString2.java)

```

1      import java.util.Scanner;
2      public class ExemploString2 {
3          public static void main(String[] args) {
4              String msg = "Aprendendo a manipular Strings";
5              int i;
6              char letra;
7              Scanner ler = new Scanner(System.in);
8              System.out.println("Digite uma letra:");
9              letra = ler.nextLine().charAt(0);
10             for (i=0; i<msg.length(); i++)
11                 if (msg.charAt(i)==letra){
12                     System.out.println("Letra encontrada na posicao "+i);
13                     break;
14                 }
15             if (i==msg.length())
16                 System.out.println("Letra não existe na mensagem!");
17         }
18     }

```

Métodos para conversão: maiúsculos e/ou minúsculos

Observações:

O método `charAt()` pode ser usado para identificar, por exemplo, se um determinado caractere da mensagem é minúsculo. Veja:



```
If ( msg.charAt(i) >= 'a' && msg.  
charAt(i) <= 'z')
```

Isso também se aplica aos caracteres maiúsculos.

A classe `String` possui métodos para converter todos os caracteres de uma *string* para maiúsculo ou para minúsculo. O método usado para converter para maiúsculos é o `toUpperCase()`, que faz com que a *string* que o referenciou seja convertida dessa forma. Essa operação deve ser atribuída a outra *string* (caso seja a própria, ela ficará toda em maiúsculos). O método equivalente que faz a conversão para minúsculo é `toLowerCase()`. O quadro 34 ilustra um exemplo em que uma *string* é convertida para maiúsculos.

Quadro 34: Exemplo de `toUpperCase()` com *strings* (arquivo: `ExemploString3.java`)

```
1 public class ExemploString3 {  
2     public static void main(String[] args) {  
3         String msg = "Aprendendo a manipular Strings";  
4         msg = msg.toUpperCase();  
5         System.out.println("Frase: "+msg);  
6     }  
7 }
```

Comparação entre *strings*

Como visto anteriormente, o operador relacional `==` deve ser usado para testar igualdade entre valores, porém, com *strings*, isso é um pouco diferente. Como elas são objetos, se usarmos o operador `==`, estaremos comparando as referências à memória (endereços) e, portanto, caso tenhamos duas *strings* referenciando o mesmo endereço, essa comparação será *true*. Isso pode até provocar um erro lógico no programa, principalmente quando criamos *strings* com inicializações.

Se inicializamos uma *string* com literais, a JVM cria um *pool* com tal informação e, ao criar outra *string* com mesmo conteúdo, procura no *pool* se existe o valor. Caso exista, atribui a nova *string* para o mesmo conteúdo, e, nesse caso, o operador `==` retornará com *true*.

Caso uma das *strings* seja instanciada com o operador `new` e nela atribuído o mesmo conteúdo, por ter sido criada uma nova referência (novo endereço de memória), o operador `==` retornará *false*. Veja exemplo no quadro 35.

Quadro 35: Comparação errônea entre *strings* (arquivo: `ExemploString4.java`)

```

1  public class ExemploString4 {
2      public static void main(String[] args) {
3          String s1 = "Java";
4          String s2 = "Java";
5          String s3;
6          s3 = new String ("Java"); // nova instância para s3 com mesmo conteudo
7          if (s1==s2) // Este teste será true - mesmos endereços
8              System.out.println("Strings iguais: "+s1+" == "+s2);
9          if (s1==s3) // este teste será false
10             System.out.println("Strings iguais: "+s1+" == "+s3);
11         else
12             System.out.println("Strings diferentes: "+s1+" != "+s3);
13     }
14 }

```

Portanto, **não** se deve usar o operador == para comparar duas *strings*.

Devemos usar o método *equals*, da superclasse *Object*, para efetuar a comparação entre duas *strings*. O exemplo do quadro 36 mostra uma comparação correta.

Quadro 36: Comparação correta entre *strings* (arquivo: ExemploString5.java)

```

1  public class ExemploString5 {
2      public static void main(String[] args) {
3          String s1 = "Java", s3;
4          s3 = new String ("Java"); // nova instancia para s3 com mesmo conteudo
5          if (s1.equals(s3)) // Este teste será true --> comparação correta
6              System.out.println("Strings iguais: "+s1+" == "+s3);
7      }
8  }

```

Observações:

- 1) Na classe *String*, existe um método chamado *equalsIgnoreCase(String)*, que compara duas *strings* ignorando se os caracteres estão em maiúsculos ou minúsculos. Veja exemplo no quadro 36.
- 2) Para saber mais sobre outros métodos, veja documentação completa em: <http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html>.



Quadro 37: Comparação entre *strings* com *equalsIgnoreCase* (arquivo: ExemploString6.java)

```

1  public class ExemploString6 {
2      public static void main(String[] args) {
3          String s1 = "JAVA", s3;
4          s3 = new String ("Java"); // nova instância para s3 com conteúdo em minúsculo

```

```

5         if (s1.equalsIgnoreCase(s3)) // Este teste será true --> comparação correta
6             System.out.println("Strings iguais: "+s1+" == "+s3);
7     }
8 }

```

Exercícios do Módulo 3

- 1) Elabore um programa em Java que leia um valor inteiro de 20 elementos e crie outro vetor, também com 20 elementos, com os mesmos valores do vetor lido, porém em ordem invertida.
- 2) Elabore um programa em Java que gere dois vetores (A e B) aleatórios de 10 valores inteiros cada e, em seguida, calcule e mostre o produto escalar entre eles. A fórmula para o produto escalar entre os vetores A e B é: $PE = A[0]*B[0] + A[1]*B[1] + A[2]*B[2] + \dots + A[9]*B[9]$
- 3) Elabore um programa em Java que gere um vetor de 50 elementos inteiros de forma aleatória (valores entre 0 e 99) e, em seguida, leia um determinado valor pelo teclado e verifique se ele existe ou não no vetor. Emita a mensagem correspondente.
- 4) Elabore um programa em Java que atribua a um vetor de 20 elementos os 20 primeiros elementos da série de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89... o próximo é a soma dos dois anteriores. Mostre o vetor gerado.
- 6) Elabore um programa em Java que gere dois vetores A e B com 10 elementos cada, contendo valores aleatórios entre 0 e 9. Em seguida, crie um terceiro vetor C que seja a *interseção* entre os vetores A e B. O conjunto interseção é formado por todos os elementos que existem tanto no vetor A quanto no B.
- 7) Elabore um programa em Java que gere dois vetores A e B com 10 elementos cada, contendo valores aleatórios entre 0 e 9. Em seguida, crie um terceiro vetor C que seja a *união* entre os vetores A e B. O conjunto união é formado por todos os elementos que existem no vetor A mais os que estão em B que não existem em A.
- 8) Elabore um programa em Java que gere uma matriz quadrada 5x5 com valores aleatórios e em seguida mostre os elementos do triângulo superior direito (elementos que estão acima da diagonal principal).
- 9) Elabore um programa em Java que gere uma matriz quadrada 5x5 com valores aleatórios e em seguida mostre os elementos do triângulo superior esquerdo (elementos acima da diagonal secundária).
- 10) Elabore um programa em Java que gere uma matriz 3x4 com valores aleatórios e calcule e mostre a matriz transposta correspondente.
- 11) Elabore um programa em Java que gere uma matriz 5x5 e verifique se ela é ou não simétrica. Uma matriz, para ser simétrica, deve ter $a[i][j] == a[j][i]$ para todo i e j.
- 12) Elabore um programa em Java que leia várias *strings* (use . para finalizar) e remova de cada *string* todas as ocorrências de uma dada letra oferecida pelo usuário.
- 13) Elabore um programa em Java que leia várias *strings* (use . para finalizar) com o nome completo de uma pessoa e em seguida crie uma *string* contendo apenas as iniciais de cada nome. Exemplo: com o nome "Jose Antonio Santos", o programa produzira "J.A.S."

Módulo 4: Pesquisa e ordenação

Este módulo 4 apresenta alguns métodos consagrados, disponíveis na literatura técnica da área de computação, que realizam a pesquisa de informações em estruturas de dados, bem como métodos para realizar a ordenação de vetores. Vamos conhecer a implementação dessas técnicas em Java. Bom estudo!

Aula 9: Métodos de ordenação

Nesta aula, são apresentados alguns métodos para ordenação de vetores. Existem muitos algoritmos na literatura, alguns com desempenhos excelentes, como QuickSort, MergeSort e HeapSort, e outros mais modestos, porém interessantes, que merecem nossa atenção. Os métodos mais avançados utilizam algoritmos recursivos e, por não abordarmos em nossa disciplina o conceito de métodos e parâmetros, não serão apresentados. Aqui vamos conhecer os algoritmos de ordenação *selection sort*, *bubble sort* e *insertion sort*. Vamos entender como esses métodos funcionam e como é sua implementação em Java.

Método de seleção (*selection sort*)

Nesse método, é selecionado o menor elemento do vetor e colocado na primeira posição (0). Em seguida, considerando os elementos restantes, é selecionado o próximo menor e colocado na segunda posição. Esse processo repete-se até que não haja mais elementos a ser selecionados, e, então, o vetor estará ordenado. A figura 9 ilustra esse algoritmo, e o quadro 37 mostra o respectivo código-fonte.

Passos

0	10	50	20	30	90
	Segmento Ordenado	Segmento Desordenado			
1	10	50	20	30	90
2	10	20	50	30	90
3	10	20	30	50	90
4	10	20	30	50	90

Esquema de ordenação por seleção. Fonte: <http://albertocn.sytes.net/2010-2/ed1/aulas/classificacao.htm>.

Quadro 38: Método seleção (arquivo: ExemploOrdenacao1.java)

```

1  import java.util.Random;
2  public class ExemploOrdenacao1 {
3      public static void main(String[] args) {
4          int a[] = new int[100];
5          int i, j, menor, posmenor;
6          boolean troca;

```



```

7      Random gerador = new Random();
8      System.out.println("Vetor Gerado: ");
9      for (i=0; i<a.length; i++){
10         a[i] = gerador.nextInt(100); // gerando um vetor aleatório
11         System.out.print(a[i]+" ");
12     }
13     for (i=0; i<a.length; i++) {
14         menor = a[i];
15         posmenor = i;
16         for (j=i+1; j<a.length; j++){
17             if (a[j] < menor) {
18                 menor = a[j];
19                 posmenor = j;
20             }
21         }
22         a[posmenor] = a[i];
23         a[i] = menor;
24     }
25     System.out.println("\nVetor ordenado: ");
26     for (i=0; i<a.length; i++)
27         System.out.print(a[i]+" ");
28 }
29 }

```

Método da bolha (*bubble sort*)

O *bubble sort* é um dos mais conhecidos e simples métodos de ordenação, mas é o que apresenta um dos piores desempenhos. Pertence à classe de métodos de ordenação baseados em permutação ou trocas. Consiste em comparar cada elemento com o próximo ($a[i]$ com $a[i+1]$) e trocá-los sempre que estiverem fora de ordem. Esse processo repete-se até que não haja mais trocas a ser realizadas e, nesse momento, o vetor estará ordenado.

Ao compararmos o bolha com o seleção, podemos notar que a estratégia do seleção é colocar os menores elementos no início e o do bolha é colocar os maiores no final do vetor.

A tabela a seguir ilustra o esquema do algoritmo bolha, e o quadro 38, o respectivo código.

Passos

0	50	40	30	20	10
1	40	30	20	10	50
2	30	20	10	40	50
3	20	10	30	40	50
4	10	20	30	40	50
5	10	20	30	40	50

Esquema de ordenação por bolha. Fonte: <http://albertocn.sytes.net/2010-2/ed1/aulas/classificacao.htm>.

Em implementações do método da bolha, é criada uma variável denominada bolha, que é usada para marcar a última posição do vetor na qual ocorreu uma troca. No próximo passo, o vetor é percorrido da posição 1 até a posição bolha. Isso pode conferir uma certa agilidade ao método, dependendo de como os elementos do vetor estejam posicionados inicialmente. No exemplo, veja o comportamento do método a partir da configuração inicial. Em dois passos, o vetor já ficou ordenado.

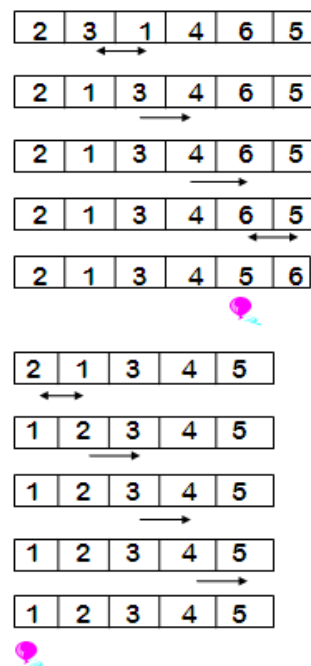
Quadro 39: Método bolha (arquivo: ExemploOrdenacao2.java)

```

1  import java.util.Random;
2  public class ExemploOrdenacao2 {
3      public static void main(String[] args) {
4          int a[] = new int[100];
5          int i, j, aux;
6          boolean troca;
7          Random gerador = new Random();
8          System.out.println("Vetor Gerado: ");
9          for (i=0; i<a.length; i++){
10             a[i] = gerador.nextInt(100); // gerando um
11             vetor aleatório
12             System.out.print(a[i]+" ");
13         }
14         j=99;
15         do
16         {
17             troca = false;
18             for (i=0; i<j; i++){
19                 if (a[i] > a[i+1]) {
20                     aux = a[i];
21                     a[i] = a[i+1];
22                     a[i+1]=aux;
23                     troca = true;
24                 }
25             }
26             j--i
27         }while(troca);
28         System.out.println("\nVetor ordenado: ");
29         for (i=0; i<a.length; i++)
30             System.out.print(a[i]+" ");
31     }

```

Fonte: Produzido pela profa. Elisabete de Oliveira



No programa apresentado no quadro 39, foi introduzida uma variável booleana, *troca*, para que não haja repetições desnecessárias. Assim, caso haja uma iteração do programa sobre o vetor e nenhuma troca tenha sido realizada, isso significa que ele já está ordenado, e o algoritmo pode terminar. Essa variável realiza a

mesma função da variável *bolha*, explicada anteriormente.

Método inserção direta (*insertion sort*)

Para um vetor com n elementos, inicia-se, a partir do segundo, comparando-o com o primeiro. O método garante que, em cada passo, todos os elementos anteriores ao que está sendo ordenado já estão em ordem.



Assim, o algoritmo de inserção direta tem o seguinte comportamento: inicialmente, compara-se o segundo elemento com o primeiro. O segundo é inserido na sua posição correta em relação ao primeiro, resultando nas duas primeiras posições ordenadas.

A seguir, o terceiro elemento é inserido na sua posição correta em relação aos anteriores, resultando nas três primeiras posições ordenadas, e assim sucessivamente. Ao inserir cada novo elemento, deslocamentos são feitos apenas se forem necessários. Imagine um jogo de baralho em que você busca ordenar as cartas em sequência deixando ordenados os elementos a cada nova carta coletada, encaixando-a na posição correta.

A tabela a seguir ilustra uma ordenação a do algoritmo de inserção.

Passos	Segmento Ordenado	Segmento Desordenado				
1	10	50	20	30	90	
2	10	50	20	30	90	
3	10	20	50	30	90	
4	10	20	30	50	90	
5	10	20	30	50	90	

Esquema d e ordenação por inserção direta. Fonte: <http://albertocn.sytes.net/2010-2/ed1/aulas/classificacao.htm>.

O quadro 40 ilustra o algoritmo inserção direta para um vetor gerado aleatoriamente com 100 elementos inteiros.

Quadro 40: Inserção direta (arquivo: ExemploOrdenacao3.java)

```

1  import java.util.Random;
2  public class ExemploOrdenacao3 {
3      public static void main(String[] args) {
4          int a[] = new int[100];
5          int i, j, aux;
6          Random gerador = new Random();
7          System.out.println("Vetor Gerado: ");
8          for (i=0; i<a.length; i++){
9              a[i] = gerador.nextInt(100); // gerando um vetor aleatório
10             System.out.print(a[i]+" ");
11         }
12         for (i=1; i<a.length; i++){
13             aux = a[i];
14             j = i-1;
15             while ((j>=0)&&(aux<a[j])) {

```

```
16         a[j+1] = a[j]; // deslocando os elementos
17         j--;
18     }
19     a[j+1]= aux; // inserindo a chave no local correto
20 }
21 System.out.println("\nVetor ordenado: ");
22 for (i=0; i<a.length; i++)
23     System.out.print(a[i]+" ");
24 }
25 }
```

Exercício

O algoritmo de ordenação *shake sort* é uma versão melhorada do *bubble sort*. No *bubble sort*, a cada iteração, o maior valor desloca-se para o seu devido lugar. No *shake sort*, ocorrem deslocamentos em ambas as extremidades, isto é, à medida que o maior se desloca para o fim do vetor, o menor desloca-se para o início. Ao terminar a iteração, ambas as extremidades são atualizadas para desconsiderar os elementos já colocados. Elabore um programa em Java que implemente esse algoritmo.

Comparação entre os métodos simples de ordenação

Todos os três métodos de ordenação que estudamos aqui não podem ser considerados os melhores, pois apresentam um número de comparações da ordem de n^2 , considerando n o número de elementos

do vetor. Em notação O , esses métodos são ditos $O(n^2)$. Assim, eles devem ser usados apenas para ordenar vetores com número de elementos pequenos.

O método da bolha e o método de seleção são ambos $O(n^2)$. Mas o número de comparações no método da bolha, para casos em que o vetor está em ordem crescente ou ordenado, é menor que no de seleção.

O método da bolha apresenta uma enorme desvantagem que o torna o de pior desempenho entre os três estudados: ele exige um grande número de trocas. Entre o método de inserção e o de seleção, o primeiro é superior em relação ao número médio de comparações.

Portanto, não existe, entre esses três métodos, um que seja bastante superior, mas apenas um que seja mais lento (o da bolha).

Aula 10: Métodos de pesquisa

Um dos principais propósitos da ordenação de vetores é facilitar a busca futura de elementos no conjunto classificado. O processo de pesquisa representa a localização de uma dada chave numa estrutura que, se estiver ordenada, permitirá a execução do trabalho em um menor tempo. O critério mais importante no julgamento da eficiência de uma técnica de pesquisa é quanto tempo é gasto para se encontrar um elemento. Como esse critério é dependente de máquina, a *medida de eficiência* de um algoritmo de pesquisa é calculada pelo *número médio de comparações* necessárias para encontrar cada elemento da lista, denotado por L (*length*) ou *comprimento médio de busca*.

Pesquisa sequencial

Ao nos depararmos com a necessidade de realizar uma pesquisa sobre um vetor, a pesquisa sequencial é o primeiro algoritmo que vem em mente, pois é o método mais simples e mais intuitivo. Para localizar uma determinada chave, ele examina sequencialmente os elementos do vetor até que ela seja localizada ou terminarem os elementos. O quadro 41 ilustra esse algoritmo.

Quadro 41: Pesquisa sequencial (arquivo: ExemploPesquisa1.java)

```

1      import java.util.Arrays;
2      import java.util.Random;
3      import java.util.Scanner;
4      public class ExemploPesquisa {
5          public static void main(String[] args) {
6              int a[] = new int[100];
7              int i, chave;
8              Random gerador = new Random();
9              Scanner ler = new Scanner(System.in);
10             for (i=0; i<a.length; i++)
11                 a[i] =gerador.nextInt(100); // gerando um vetor aleatorio
12             System.out.println("Digite a chave para busca: ");
13             chave = ler.nextInt();
14             for (i=0; i<a.length; i++){
15                 if (a[i] == chave) {
16                     System.out.println("Elemento encontrado na posicao: "+i);
17                     break;
18                 }
19             }
20             if (i==a.length)
21                 System.out.println("Elemento nao existe no vetor!");
22         }
23     }

```

Percebe-se que, nessa pesquisa, o pior caso realizará n comparações (n é o tamanho do vetor). O desempenho médio desse algoritmo é dado por $(n + 1)/2$. Uma forma de melhorá-lo é fazer com que essa busca sequencial seja realizada sobre um vetor ordenado.

Pesquisa sequencial ordenada

Essa pesquisa é realizada sobre um vetor ordenado. Com isso, garante-se um aumento no desempenho do algoritmo, porque não haverá necessidade de pesquisarmos até o final do vetor para concluirmos que a chave não existe: bastará encontrar o primeiro valor maior que a chave que essa conclusão se torna óbvia. O quadro 41 ilustra esse algoritmo. Para ordenar o vetor gerado, usaremos a classe *Arrays*, que contém o método *sort* para fazer com que o vetor seja ordenado de forma crescente.

Quadro 42: Pesquisa sequencial ordenada (Arquivo: ExemploPesquisa2.java)

```
1  import java.util.Arrays;
2  import java.util.Random;
3  import java.util.Scanner;
4  public class ExemploPesquisa2 {
5      public static void main(String[] args) {
6          int a[] = new int[100];
7          int i,chave;
8          Random gerador = new Random();
9          Scanner ler = new Scanner(System.in);
10         for (i=0; i<a.length; i++)
11             a[i] = gerador.nextInt(100);      // gerando um vetor aleatorio
12         Arrays.sort(a);      //ordenando o vetor a de forma crescente
13         System.out.println("Digite a chave para busca: ");
14         chave = ler.nextInt();
15         i=0;
16         while (a[i] < chave && i<a.length)
17             i++;
18         if (a[i] == chave)
19             System.out.println("Elemento encontrado na posicao: "+i);
20         else
21             System.out.println("Elemento nao existe no vetor!");
22     }
23 }
```

Ainda assim, se a chave a ser pesquisada é maior que o último elemento do vetor, teremos o pior caso desse algoritmo, resultando em n comparações. Porém, para as demais pesquisas malsucedidas, ele tem um comportamento melhor que o da pesquisa sequencial.

Uma forma de melhorar ainda mais essa pesquisa é usar o algoritmo de pesquisa binária, como veremos a seguir.

Pesquisa binária

Esse algoritmo também necessita que o vetor no qual ocorrerá a busca esteja ordenado. Mas, ao invés de efetuar uma pesquisa sequencial, o método consiste na comparação da chave procurada com o elemento localizado no endereço médio do vetor ($n/2$). Se a chave procurada for menor que o elemento dessa posição (meio do vetor), o processo é repetido para a primeira metade do vetor apenas e, se for maior, para a segunda metade apenas. Se for igual, a pesquisa encerra-se bem-sucedida. Caso não haja mais metades nas quais buscar a chave, termina malsucedida.

Numa explicação mais detalhada, para se realizar a pesquisa binária, devem ser seguidos os passos:

- 1) Determinar o elemento que está no meio do vetor e compará-lo com o valor procurado (chave K). O elemento do meio está na posição $meio = (esq + dir) / 2$.
- 2) Se o elemento central for igual a K, a pesquisa termina.
- 3) Se o elemento central for menor que K, a pesquisa continuará na metade superior (a inferior será descartada), ou seja, o *dir* será mantido e o *esq*, ajustado para a posição $meio + 1$.
- 4) Já se o elemento central for maior que K, continua-se a pesquisa somente na metade inferior do vetor, ou seja, o *esq* será mantido e o *dir*, ajustado para a posição $meio - 1$.
- 5) E assim sucessivamente...

A figura a seguir ilustra esse processo.

A pesquisa se encerrará em dois casos: ou quando a chave for encontrada, ou quando não houver mais nenhum elemento do vetor a ser pesquisado. A não localização do elemento procurado ocorre quando *começo* fica maior que *fim*.

O procedimento acima descrito aplica-se a vetores classificados em ordem crescente. Para os em ordem decrescente, deve aplicar-se um raciocínio análogo.



A quantidade de comparações que o método de pesquisa binária realiza é aproximadamente igual ao número de vezes que n (número de elementos do vetor) pode ser dividido por 2 até resultar 1, isto é, $O(\log_2 n)$. Assim, a ordem de complexidade desse método é logarítmica, o que significa que é muito eficiente.

Observemos alguns valores do logaritmo base 2 de alguns números: $\log_2 8 = 3$ (para um vetor de tamanho 8, são realizadas três comparações no máximo para se localizar um elemento ou para se concluir que ele não está no vetor); $\log_2 256 = 8$ (para um vetor de tamanho 256, oito comparações no máximo); $\log_2 1024 = 10$ (para um vetor de tamanho 1.024, são realizadas 10 comparações no máximo); $\log_2 4096 = 12$ (para 4.096 elementos, apenas 12 comparações no máximo).

O quadro 43 apresenta o algoritmo do método de pesquisa binária.

Quadro 43: Pesquisa binária (arquivo: ExemploPesquisa3.java)

```

1  import java.util.Arrays;
2  import java.util.Random;
3  import java.util.Scanner;
4  public class ExemploPesquisa3 {
5      public static void main(String[] args) {
6          int a[] = new int[100];
7          int meio, esq, dir, chave, pos = -1, i;
8          Random gerador = new Random();
9          Scanner ler = new Scanner(System.in);
10         for (i = 0; i < a.length; i++)
11             a[i] = gerador.nextInt(100); // gerando um vetor aleatorio
    
```



```

12     Arrays.sort(a); //ordenando o vetor a de forma crescente
13     System.out.println("Digite a chave para busca: ");
14     chave = ler.nextInt();
15     esq=0; dir=99;
16     while (esq < dir && pos!=-1) {
17         meio=(esq+dir)/2;
18         if (a[meio]==chave)
19             pos = meio;
20         else
21             if (chave> a[meio])
22                 esq = meio+1;
23             else
24                 dir = meio-1;
25     }
26     if (pos>=0)
27         System.out.println("Elemento encontrado na posicao: "+pos);
28     else
29         System.out.println("Elemento nao existe no vetor!");
30 }
31 }

```

Exercício

Modifique o programa do quadro 43 substituindo o método de ordenação `Array.sort()` pelo algoritmo de ordenação por inserção.

Exercício do Módulo 4

Implementar em Java o método de ordenação *counting sort*, conforme explicado abaixo.

A ordenação por contagem pressupõe que cada um dos N elementos a ser ordenados é um valor inteiro entre 0 e k . A ideia básica é determinar, para cada elemento da lista de entrada, o número de elementos menores ou iguais a ele. Essa informação é usada para determinar o lugar onde cada elemento deve ser inserido na lista final (ordenada). Considere como exemplo uma lista com $N=8$ elementos a ser ordenados, menores ou iguais a $k=5$ (note que essa lista não usa a posição 0 do vetor). Veja:

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

O próximo passo é criar um vetor C , contendo $K+1$ elementos, iniciados com 0

	0	1	2	3	4	5
C	0	0	0	0	0	0

Os índices do vetor C (0 a 5) são os valores dos elementos do vetor A , e os conteúdos de suas posições, a quantidade deles no vetor A . Assim, o vetor C ficará:

	0	1	2	3	4	5
C	2	0	2	3	0	1

Após isso, o vetor C é atualizado, indicando a quantidade de elementos existentes em A que são menores ou iguais a cada um. Veja:

	0	1	2	3	4	5
C	2	2	4	7	7	8

Agora é só organizar o vetor B (vetor resposta, ordenado), examinando os elementos de A do último para o primeiro. Na medida em que um valor entra em B, o vetor C é atualizado. Quando A acabar ou quando C estiver totalmente zerado, B conterá os valores de A ordenados.

	1	2	3	4	5	6	7	8
B							3	

Para o elemento $A[8]=3$, o vetor C indica que 3 deve ser inserido na posição 7 de B. Então, insere-se 3 em B[7] e decrementa-se C[3]:

	0	1	2	3	4	5
C	2	2	4	6	7	8

Para o elemento $A[7]=0$, o vetor C indica que 0 deve ser inserido na posição 2 de B. Então, insere-se 0 em B[2] e decrementa-se C[0]:

	1	2	3	4	5	6	7	8
B	0						3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

Esse processo repete-se até que a varredura do vetor A termine. Assim, o vetor B ficará:

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

Imprime-se o vetor B, ordenado.

Considerações finais

Prezado aluno, é com satisfação que chegamos ao fim de mais uma disciplina. Espero que os conteúdos aqui ministrados tenham atingido seus objetivos e expectativas.

Como falamos no começo, o objetivo principal desta disciplina foi o de ministrar as principais estruturas de programação da linguagem Java, visando ao estudo de lógica de programação. Assim, foram apresentados desde programas básicos envolvendo estruturas sequenciais até outros mais elaborados, com vetores, matrizes e métodos de ordenação e pesquisa, que fazem uso de estruturas de dados, repetições e seleções.

A linguagem Java foi apresentada, assim como seu compilador *javac* e a forma de executar os programas com *java* via linha de comando. Muitas aplicações foram exemplificadas e também muitos exercícios foram definidos para que tais conhecimentos sejam gravados e aprendidos. Não foram poupados detalhes nas apresentações das declarações, operações e manipulações dessas estruturas de programação.

Esta disciplina servirá de suporte a outras. Assim, os conteúdos foram cuidadosamente preparados, e este material, editado para que você tenha em mãos um guia de programação simples, versátil e útil para eventuais consultas futuras.

Coloco-me à disposição para eventuais dúvidas e esclarecimentos, deixando a você meus sinceros votos de sucesso na vida e na profissão.

Prof. Mauricio Duarte

Respostas Comentadas dos Exercícios

Aula 1

Exercícios (pag. 11)

1a) md Mauricio

cd Mauricio

1b)

```
public class Exercicio_aula_1_b {  
    public static void main(String[] args) {  
        System.out.println("Mauricio Duarte");  
        System.out.println("Rua Carolina Moraes Almeida, 10 - Marilia - SP");  
        System.out.println("Tel. 14 99999 7777");  
        System.out.println("E-mail: maur.duarte@gmail.com");  
    }  
}
```

Aula 2

Exercícios (pag. 15)

1) Basta digitar o código proposto, compila-lo e executá-lo. Analise as saídas de dados.

2)

```
public class Exercicio_aula_2_2 {  
    public static void main(String[] args) {  
        double nota1, nota2, nota3, media;  
        nota1 = 5.0;  
        nota2 = 3.5;  
        nota3 = 9.5;  
        media = (nota1+nota2+nota3)/3;  
        System.out.println("Media = "+media);  
    }  
}
```

3)

```
public class Exercicio_aula_2_3 {  
    public static void main(String[] args) {  
        double preco_venda, preco_custo;  
        preco_custo = 37;  
        preco_venda = preco_custo + (preco_custo*0.12) + (preco_custo*0.2695);  
    }  
}
```

```

        System.out.println("Preço de venda: R$ "+preco_venda);
    }
}

```

Aula 3

Exercícios (pag. 19)

- 1) Apenas digitar o algoritmo abordado na seção 3.2, quadro 8, compilá-lo e executá-lo.
- 2)

```

import java.util.Scanner;
public class Exercicio_aula_3_2 {
    public static void main(String[] args) {
        int a,b,c;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um valor para A: ");
        a = ler.nextInt();
        System.out.println("Digite um valor para B: ");
        b = ler.nextInt();
        // trocando os conteudos de a com b
        c = a;
        a = b;
        b = c;
        System.out.println("Valor de A = "+a+" Valor de B = "+b);
    }
}

```

Exercícios do Módulo 1

- 1) Identificadores corretos: a) b) d) e g)
- 2) A = 11, B = 11 e C = 11
- 3)

```

import java.util.Scanner;
public class Modulo_1_exercicio_3 {
    public static void main(String[] args) {
        int a,b,c;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite o preço do produto: ");
        preco = ler.nextDouble();
        preco_desconto = preco*0.90;
        System.out.println("Preço com desconto: R$"+preco_desconto);
    }
}

```

4)

```
import java.util.Scanner;
public class Modulo_1_exercicio_4 {
    public static void main(String[] args) {
        double valor, cot, dolar;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite a cotação do dolar: ");
        cot = ler.nextDouble();
        System.out.println("Digite o valor em R$ a ser convertido: ");
        valor = ler.nextDouble();
        dolar = valor/cot;
        System.out.println("Total de dolares: $ "+dolar);
    }
}
```

5)

```
import java.util.Scanner;
public class Modulo_1_exercicio_5 {
    public static void main(String[] args) {
        int num, c, d, u;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um numero de 3 algarismos: ");
        num = ler.nextInt();
        c = num/100;
        d = (num%100)/10;
        u = num%10;
        System.out.println("Centena = "+c+" Dezena = "+d+" Unidade = "+u);
    }
}
```

6)

```
import java.util.Scanner;
public class Modulo_1_exercicio_6 {
    public static void main(String[] args) {
        int tempo, h, m, s;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um tempo em segundos: ");
        tempo = ler.nextInt();
        h = tempo/3600;
        m = (tempo % 3600)/60;
        s = tempo % 60;
        System.out.println("Horas = "+h+" Minutos = "+m+" Segundos = "+s);
    }
}
```

Aula 4

Exercícios (pag. 24)

1)

```
import java.util.Scanner;
public class Exercicio_aula_4_1 {
    public static void main(String[] args) {
        int a,b,c, maior;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um valor para A: ");
        a = ler.nextInt();
        System.out.println("Digite um valor para B: ");
        b = ler.nextInt();
        System.out.println("Digite um valor para C: ");
        c = ler.nextInt();
        if ((a>b) && (a>c))
            maior = a;
        else
            if (b>c)
                maior = b;
            else
                maior = c;
        System.out.println("Maior valor = "+ maior);
    }
}
```

2)

```
import java.util.Scanner;
public class Exercicio_aula_4_2 {
    public static void main(String[] args) {
        int ano;
        Scanner ler = new Scanner(System.in);
        Scanner ler = new Scanner(System.in);
        ano = ler.nextInt();
        if((ano%400==0) || ((ano % 4 ==0)&& (ano % 100>0)))
            System.out.println(" O ano "+ano+" é bissexto!");
        else
            System.out.println(" O ano "+ano+" NÃO é bissexto!");
    }
}
```

Aula 5

Exercícios (pag. 27)

1)

```
import java.util.Scanner;
public class Exercicio_aula_5_1 {
    public static void main(String[] args) {
        int num, soma=0, cont=1;
        Scanner ler = new Scanner(System.in);
        while (cont<=10) {
            System.out.println("Digite um numero: ");
            num = ler.nextInt();
            soma += num;
            cont++;
        }
        System.out.println("Soma = "+soma);
    }
}
```

2)

```
import java.util.Scanner;
public class Exercicio_aula_5_2 {
    public static void main(String[] args) {
        int homens=0, mulheres=0, cont=1;
        char sexo;
        Scanner ler = new Scanner(System.in);
        while (cont<=20) {
            System.out.println("Digite o sexo do aluno (m ou f): ");
            sexo = ler.nextLine().charAt(0);
            if (sexo=='m')
                homens++;
            else
                mulheres++;
            cont++;
        }
        System.out.println("Total de homens = "+homens);
        System.out.println("Total de mulheres = "+mulheres);
    }
}
```

3)

```
public class Exercicio_aula_5_3 {
    public static void main(String[] args) {
        int num=1, soma=0;
        while (true) {
            soma += num;
            ++num;
        }
    }
}
```



```

        if (num > 100)
            break;
    }
    System.out.println("Soma dos números entre 1 e 100 = "+soma);
}
}

```

Resposta: o comando **while(true)** provoca uma repetição infinita, pois será sempre verdadeiro. Porém, ao executar o comando **if(num > 100)** o comando **break** interrompe esta repetição infinita e, com isso, a variável soma possuirá o valor correto dos valores somados.

Exercícios (pag. 28)

1)

```

import java.util.Scanner;
public class Exercicio_aula_5_1_p28 {
    public static void main(String[] args) {
        int idade, cont=0;
        double soma=0, media;
        Scanner ler = new Scanner(System.in);
        do {
            System.out.println("Digite uma idade ou 0(zero) para encerrar: ");
            idade = ler.nextInt();
            if (idade > 0) {
                soma += idade;
                cont++;
            }
        }while (idade>0);
        if (cont==0)
            System.out.println("Nenhuma idade foi digitada!");
        else {
            media = soma/cont;
            System.out.println("Media entre as idades = "+media);
        }
    }
}

```

Exercícios (pag. 30)

1)

```

import java.util.Scanner;
public class Exercicio_aula_5_1_p30 {
    public static void main(String[] args) {
        int i, num, soma = 0;
        Scanner ler = new Scanner(System.in);

```

```

        for (i=1; i<=10; i++) {
            System.out.println("Digite um numero: ");
            num = ler.nextInt();
            soma += num;
        }
        System.out.println("Soma = "+soma);
    }
}

```

- 2) Basta digitar efetuando as respectivas substituições. O resultado provocado em cada algoritmo deverá ser o mesmo.

Exercícios (pag. 32)

1)

```

public class Exercicio_aula_5_1_p32 {
    public static void main(String[] args) {
        for ( int i=0; i<=300; i++) {
            if (i%3 > 0)
                continue;
            System.out.println(i);
        }
    }
}

```

Exercícios do Módulo 2

1)

```

import java.util.Scanner;
public class Modulo_2_exercicio_1 {
    public static void main(String[] args) {
        int num, d;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um numero: ");
        num = ler.nextInt();
        System.out.println("Divisores de "+num);
        for (d=1; d<=num; d++)
            if (num % d == 0)
                System.out.print(d+" ");
    }
}

```

2)

```

import java.util.Scanner;
public class Modulo_2_exercicio_2 {

```

```

public static void main(String[] args) {
    int num, d, td = 0 ; // td é o total de divisores de num
    Scanner ler = new Scanner(System.in);
    System.out.println("Digite um numero: ");
    num = ler.nextInt();
    for (d=1; d<=num; d++)
        if (num % d == 0)
            td++;
    if (td == 2) // divisores o 1 e ele próprio
        System.out.println("O numero "+num+" é primo!");
    else
        System.out.println("O numero "+num+" Não é primo!");
}
}

```

3)

```

public class Modulo_2_exercicio_3 {
    public static void main(String[] args) {
        int num, soma=0;
        for (num=1; num<=100; soma+=num, num++);
        System.out.println("A soma 1+2+3+4+...+100 = "+soma);
    }
}

```

4)

```

import java.util.Scanner;
public class Modulo_2_exercicio_4 {
    public static void main(String[] args) {
        double a,b,resp=0;
        boolean operacao;
        char op;
        Scanner ler = new Scanner(System.in);
        do {
            System.out.println("Digite uma operação (+, -, *, / ou # para encerrar: ");
            op = ler.nextLine().charAt(0);
            if (op!='#') {
                operacao = true;
                System.out.println("Digite um numero: ");
                a = ler.nextDouble();
                System.out.println("Digite outro numero: ");
                b = ler.nextDouble();
                switch(op){
                    case '+': resp = a+b; break;
                    case '-': resp = a-b; break;
                    case '*': resp = a*b; break;
                    case '/': {

```

```

        if(b!=0)
            resp = a/b;
        else
            operacao = false;
        break;
    }
}
if (operacao)
    System.out.println("Resultado =" + resp);
else
    System.out.println("ERRO: divisao por zero!");
ler.nextLine(); // esvaziando o buffer do teclado
}
}while (op!='#');
}
}

```

5)

```

import java.util.Scanner;
public class Modulo_2_exercicio_5 {
    public static void main(String[] args) {
        double peso, alt, imc;
        int cont=1;
        Scanner ler = new Scanner(System.in);
        for (cont=1; cont<=3; cont++) {
            System.out.println("Digite o peso da pessoa: ");
            peso = ler.nextDouble();
            System.out.println("Digite a altura da pessoa: ");
            alt = ler.nextDouble();
            imc = peso/(alt*alt);
            if (imc < 18)
                System.out.println("Abaixo do peso! IMC = "+imc);
            else
                if (imc < 25)
                    System.out.println("Peso Normal! IMC = "+imc);
                else
                    if (imc < 30)
                        System.out.println("Acima do peso! IMC = "+imc);
                    else
                        System.out.println("Obesidade! IMC = "+imc);
        }
    }
}

```

6)

```
import java.util.Scanner;
public class Modulo_2_exercicio_6 {
    public static void main(String[] args) {
        int a,b,c, aux;
        char resp;
        Scanner ler= new Scanner(System.in);
        do {    System.out.println("Digite o valor de A:");
            a = ler.nextInt();
            System.out.println("Digite o valor de B:");
            b = ler.nextInt();
            System.out.println("Digite o valor de C:");
            c = ler.nextInt();
            if ((a>b)|| (a>c)) {
                if (b<c) {
                    aux = a;
                    b = aux;
                    a = b;
                }
                else {
                    aux = a;
                    a = c;
                    c = aux;
                }
            }
            if (b>c) {
                aux = b;
                b = c;
                c = aux;
            }
            System.out.println("A = "+a+" B = "+b+" C = "+c);
            ler.nextLine(); // limpando o buffer do teclado
            System.out.println("Deseja executar novamente? (s ou n)");
            resp = ler.nextLine().charAt(0);
        } while (resp=='s');
    }
}
```

7)

```
public class Modulo_2_exercicio_6 {
    public static void main(String[] args) {
        int num, p1, p2, soma; // p1: 2 primeiros e p2: os ultimos algarismos
        for (num = 1000; num<10000; num++){
            p1 = num/100;
            p2 = num%100;
```

```

        soma = p1+p2;
        if (soma*soma== num)
            System.out.println(num);
    }
}

```

8)

```

public class Modulo_2_exercicio_8 {
    public static void main(String[] args) {
        double soma = 0, numerador = 1;
        for ( int i=1; i<=20; i++) {
            soma += numerador/i;
            numerador *= 2;
        }
        System.out.println("Soma = "+soma);
    }
}

```

Aula 6

Exercícios (pag. 35)

1)

```

import java.util.Scanner;
public class Modulo_2_exercicio_5 {
    public static void main(String[] args) {
        int a [] = new int[10]; // declarando um vetor a com 5 elementos
        int i, maior, menor;
        Scanner ler = new Scanner(System.in);
        for (i=0; i<10; i++) {
            System.out.print( "A ["+i+"] = ");
            a[i] = ler.nextInt(); // lendo cada elementos em sua devida posição
        }
        maior = menor = a[0]; // inicializando maior e menor com o primeiro elemento
        for (i=1; i<10; i++) {
            if (a[i] > maior)
                maior = a[i];
            else
                if (a[i] < menor)
                    menor = a[i];
        }
        System.out.println("Mostrando quem é o maior e o menor...:");
        for (i=0; i<10; i++){
            if (a[i]==menor)
                System.out.println("A ["+i+"] = "+a[i]+" <-- Menor");
        }
    }
}

```

```

        else
            if (a[i]==maior)
                System.out.println("A ["+i+"] = "+a[i]+" <-- Maior");
            else
                System.out.println("A ["+i+"] = "+a[i]);
        }
    }
}

```

2)

```

import java.util.Scanner;
public class Exercicio_aula_6_2 {
    public static void main(String[] args) {
        float notas[] = new float[10];
        int i;
        float soma=0f, media;
        Scanner ler = new Scanner(System.in);
        for (i=0; i<10; i++) {
            System.out.print( "Nota ["+i+"] = ");
            notas[i] = ler.nextFloat();
        }
        for (i=1; i<10; i++)
            soma += notas[i]; // somando todas as notas
        media = soma/10;
        System.out.println("Media da Classe =" +media);
    }
}

```

Aula 7

Exercícios (pag. 41)

1)

```

import java.util.Scanner;
public class Exercicio_aula_7_1 {
    public static void main(String[] args) {
        int a[][] = new int [5][5];
        int i, j;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite os elementos da Matriz 5x5:");
        for (i=0; i<5; i++){
            for (j=0; j<5; j++){
                a[i][j] = ler.nextInt( );
            }
        }
        System.out.println("Diagonal Secundária: ");
        for (i=0, j=4; i<5; i++, j--){
            System.out.println( a[i][j]);
        }
    }
}

```

```
    }
}
}
```

2)

```
import java.util.Scanner;
public class Modulo_2_exercicio_5 {
    public static void main(String[] args) {
        int a[][] = new int [5][5];
        int soma[] = new int [5];
        int i, j;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite os elementos da Matriz 5x5:");
        for (i=0; i<5; i++){
            for (j=0; j<5; j++){
                a[i][j] = ler.nextInt( );
            }
            for (i=0; i<5; i++){
                soma [i]=0;
                for (j=0; j<5; j++){
                    soma[i] +=a[i][j];
                }
            }
            System.out.println ("\nMatriz lida ..... Vetor Soma");
            for (i=0; i<5; i++) {
                for (j=0; j<5; j++)
                    System.out.print(a[i][j]+" ");
                System.out.println("_____"+ soma[i]);
            }
        }
    }
}
```

Exercícios do Módulo 3

1)

```
import java.util.Random;
public class Modulo_3_Exercicio_1 {
    public static void main(String[] args) {
        int a [] = new int[10], i, aux;
        Random gerador = new Random();
        System.out.println("Gerando elementos aleatorios no vetor:");
        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(10)+10;
            System.out.print("A["+i+"]="+a[i]+" ");
        }
    }
}
```



```

    }
    // invertendo o vetor...
    for (i=0; i < a.length/2 ; i++){ // trocar metade dos elementos
        aux = a[i];
        a[i] = a[a.length-1-i];
        a[a.length-1-i]= aux;
    }
    System.out.println("\nVetor Invertido:");
    for (i=0; i<a.length; i++)
        System.out.print("A["+i+"]="+a[i]+" ");
}
}

```

2)

```

import java.util.Random;
public class Modulo_3_Exercicio_2 {
    public static void main(String[] args) {
        int a [] = new int[10];
        int b [] = new int[10];
        int i, pe=0;
        Random gerador = new Random();
        System.out.println("Gerando elementos dos vetores A e B...");
        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(10)+10;
            b[i] = gerador.nextInt(10)+10;
        }
        for (i=0; i < a.length ; i++)
            pe += (a[i]*b[i]);
        System.out.println("\nProduto Escalar: "+pe);
    }
}

```

3)

```

import java.util.Random;
import java.util.Scanner;
public class Modulo_3_Exercicio_3 {
    public static void main(String[] args) {
        int a [] = new int[50];
        int i, val;
        boolean existe = false;
        Random gerador = new Random();
        Scanner ler = new Scanner(System.in);
        System.out.println("Gerando elementos do vetor A ...");
        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(100);
            System.out.print("A["+i+"]="+a[i]+" ");

```

```

    }
    System.out.print("\nDigite um valor para consulta: ");
    val = ler.nextInt();
    for (i=0; i < a.length ; i++){
        if (a[i]==val) {
            existe = true;
            break;
        }
    }
    if (existe)
        System.out.println("\nO valor "+val+" existe no vetor!");
    else
        System.out.println("\nO valor "+val+" NAO existe no vetor!");
}
}

```

4)

```

public class Modulo_3_Exercicio_4 {
    public static void main(String[] args) {
        int a [] = new int[20];
        int a [] = new int[20];
        a[0]=a[1]=1; // inicializando os dois primeiros com 1
        for (i=2; i<a.length; i++){
            a[i] = a[i-1]+a[i-2];
        }
        System.out.println("20 primeiros numeros de Fibonacci... ");
        for (i=0; i < a.length ; i++)
            System.out.print("A["+i+"]="+a[i]+" ");
    }
}

```

5)

```

import java.util.Scanner;
public class Modulo_3_Exercicio_5 {
    public static void main(String[] args) {
        int a[] = new int[10];
        int b[] = new int[10];
        int c[] = new int[10];
        int i,j,k;
        Random gerador = new Random();
        System.out.println("Gerando elementos dos vetor A...");
        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(50);
            System.out.print("A["+i+"] = "+a[i]+" ");
        }
        System.out.println("\n\nGerando elementos dos vetor B...");
    }
}

```

```

        for (i=0; i<b.length; i++){
            b[i] = gerador.nextInt(50);
            System.out.print("B["+i+"] = "+b[i]+" ");
        }
        // calculando o vetor C - interseção de A com B
        for (k=0, i=0; i < a.length ; i++){
            for (j=0; j<b.length; j++){
                if (a[i]==b[j])
                    c[k++]=a[i];
            }
        }
        if (k==0)
            System.out.println("\n\nInterseção Vazia!");
        else
        {
            System.out.println("\n\nInterseção de A com b:");
            for (i=0; i<k; i++)
                System.out.print("C["+i+"] = "+c[i]+" ");
        }
    }
}

```

6)

```

import java.util.Random;
public class Modulo_3_Exercicio_6 {
    public static void main(String[] args) {
        int a[] = new int[10];
        int b[] = new int[10];
        int c[] = new int[20]; // uniao de A com B pode gerar 20 elementos
        int i,j,k;
        boolean existe;
        Random gerador = new Random();
        System.out.println("Gerando elementos dos vetor A...");
        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(50);
            System.out.print("A["+i+"] = "+a[i]+" ");
            c[i] = a[i]; // copiando elementos de A para C
        }
        System.out.println("\n\nGerando elementos dos vetor B...");
        for (i=0; i<b.length; i++){
            b[i] = gerador.nextInt(50);
            System.out.print("B["+i+"] = "+b[i]+" ");
        }
        // calculando o vetor C - união de A com B
        for (k=a.length,j=0; j < b.length ; j++){

```

```

        existe = false;
        for (i=0; i<a.length; i++){
            if (a[i]==b[j]) {
                existe = true;
                break;
            }
        }
        if (!existe)
            c[k++]=b[j];
    }
    System.out.println("\n\nUNIAO de A com b:");
    for (i=0; i<k; i++)
        System.out.print("C["+i+"] = "+c[i]+" ");
}
}

```

7)

```

import java.util.Random;
public class Modulo_3_Exercicio_7 {
    public static void main(String[] args) {
        int a[][] = new int[5][5];
        int i,j;
        Random gerador = new Random();
        System.out.println("Gerando elementos aleatorios da matriz...");
        for (i=0; i<5; i++){
            for (j=0; j<5; j++)
                a[i][j] = gerador.nextInt(80)+10;
        }
        System.out.println("Triangulo Superior Direito...");
        for (i=0; i<5; i++) {
            for (j=0; j<5; j++)
                if(i<j)
                    System.out.print(a[i][j]+" ");
                else
                    System.out.print(" ");
            System.out.println();
        }
    }
}

```

8)

```

import java.util.Random;
public class Modulo_3_Exercicio_8 {
    public static void main(String[] args) {
        int a[][] = new int[5][5];
        int i,j;
    }
}

```

```

Random gerador = new Random();
System.out.println("Gerando elementos aleatorios da matriz...");
for (i=0; i<5; i++){
    for (j=0; j<5; j++)
        a[i][j] = gerador.nextInt(80)+10;
}
System.out.println("Triangulo Superior Esquerdo...");
for (i=0; i<5; i++) {
    for (i=0; i<5; i++) {
        if(i+j<5)
            System.out.print(a[i][j]+" ");
        else
            System.out.print(" ");
    }
}
}

```

9)

```

import java.util.Random;
public class Modulo_3_Exercicio_9 {
    public static void main(String[] args) {
        int a[][] = new int[3][4]; // matriz de origem
        int t[][] = new int[4][3]; // matriz transposta
        int i,j;
        Random gerador = new Random();
        System.out.println("Gerando elementos aleatorios da matriz...");
        for (i=0; i<3; i++){
            for (j=0; j<4; j++) {
                a[i][j] = gerador.nextInt(80)+10;
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
        // calculando a matriz transposta
        for (i=0; i<4; i++)
            for (j=0; j<3; j++)
                t[i][j] = a[j][i]; // linha de A é coluna de T e coluna de A é
linha de T
        System.out.println("Matriz Transposta...");
        for (i=0; i<4; i++) {
            for (j=0; j<3; j++)
                System.out.print(t[i][j]+" ");
            System.out.println();
        }
    }
}

```

}

10)

```
import java.util.Random;
public class Modulo_3_Exercicio_10 {
    public static void main(String[] args) {
        int a[][] = new int[5][5];
        boolean simetria = true;
        int i,j;
        Random gerador = new Random();
        System.out.println("Gerando elementos aleatorios da matriz...");
        for (i=0; i<5; i++){
            for (j=0; j<5; j++) {
                a[i][j] = gerador.nextInt(10)+10;
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
        // verificando se a matriz A é simétrica
        for (i=0; i<5; i++)
            for (j=0; j<5; j++)
                if (a[i][j] != a[j][i]) { // invalida a simetria
                    simetria = false;
                    break;
                }
        if (simetria)
            System.out.println("Matriz é Simétrica!");
        else
            System.out.println("Matriz não é Simétrica!");
    }
}
```

11)

```
import java.util.Scanner;
public class Modulo_3_Exercicio_11 {
    public static void main(String[] args) {
        String str, strnova="";
        char letra;
        int i;
        Scanner ler = new Scanner(System.in);
        do
        {
            strnova = "";
            System.out.println("Digite uma frase ou ponto(.) para encerrar: ");
            str = ler.nextLine();
            if (str.equals("."))
```

```

        break;
        System.out.println("Digite a letra a ser removida: ");
        letra = ler.nextLine().charAt(0);
        for (i=0; i<str.length();i++) {
            if (str.charAt(i)!=letra)
                strnova += str.charAt(i);
        }
        System.out.println("String final = "+strnova);
    } while (true);
}
}

```

12)

```

import java.util.Scanner;
public class Modulo_3_Exercicio_12 {
    public static void main(String[] args) {
        String nome, sigla="";
        char letra;
        int i;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um nome completo: ");
        nome = ler.nextLine();
        sigla += nome.charAt(0);
        // copiando as iniciais e ignorando "de", "dos", "e"
        for (i=0; i<nome.length();i++) {
            if ((nome.charAt(i)==' ') && (nome.charAt(i+1)<='Z')){
                sigla += '.';
                sigla += nome.charAt(i+1);
            }
        }
        sigla += '.';
        System.out.println("Abreviação = "+sigla);
    }
}

```

Aula 9

Exercício (pag. 52)

```

import java.util.Random;
public class Exercicio_aula_10_4 {
    public static void main(String[] args) {
        int a[] = new int[100];
        int i, j, inicio, fm, aux;
        boolean troca;
        Random gerador = new Random();
        System.out.println("Vetor Gerado: ");
    }
}

```

```

        for (i=0; i<a.length; i++){
            a[i] = gerador.nextInt(100); // gerando um vetor aleatorio
            System.out.print(a[i]+" ");
        }
        fim=99;
        inicio=0;
        do
        { troca = false;
            for (i=0; i<fim; i++){
                if (a[i] > a[i+1]) {
                    aux = a[i];
                    a[i] = a[i+1];
                    a[i+1]=aux;
                    troca = true;
                }
            }
            fim--;
            for(i=fim; i>inicio; i--) {
                if (a[i] < a[i-1]) {
                    aux = a[i];
                    a[i] = a[i-1];
                    a[i-1] = aux;
                    troca = true;
                }
            }
            inicio++;
        }while((inicio<=fim) &&(troca));
        System.out.println("\nVetor ordenado por ShakeSort: ");
        for (i=0; i<a.length; i++)
            System.out.print(a[i]+" ");
    }
}

```

Aula 10

Exercícios (pag. 56)

Resposta: Basta retirar a linha `Array.sort()` e, em seu lugar, copiar as linhas entre 13 e 20, do quadro 39.

Exercício do Módulo 4

```

import java.util.Random;

public class Counting_sort {

```



```

public static void main(String[] args) {
    int a[] = new int[11];
    int b[] = new int[11];
    int c[] = new int[11];
    int i;
    Random gerador = new Random();
    System.out.println("Vetor Gerado: ");
    for (i=0; i<a.length; i++){
        a[i] = gerador.nextInt(10); // gerando um vetor aleatorio
        System.out.print(a[i]+" ");
        c[i] = 0; // inicializando o vetor c com zeros
    }
    // contando a para o vetor c
    for (i=1; i<a.length; i++)
        c[a[i]]++;
    //atualizando o vetor c com os somatorios anteriores
    for (i=1; i<c.length; i++)
        c[i] += c[i-1];
    // ordenado em b
    for (i=a.length-1; i>0; i--) {
        b[c[a[i]]] = a[i];
        c[a[i]]--;
    }
    System.out.println("\nVetor ordenado por Counting Sort: ");
    for (i=1; i<b.length; i++)
        System.out.print(b[i]+" ");
}
}

```

Referências Bibliográficas

COSTA NETO, Alberto. **Métodos de classificação**. Disponível em: <http://albertocn.sytes.net/2010-2-ed1/aulas/classificacao.htm>, acesso em: 3.mar.2014.

DEITEL, H.; DEITEL, P. **Java: como programar**. 6ª ed. São Paulo: Prentice-Hall, 2005.

FLANAGAN, D. **Java: o guia essencial**. Porto Alegre: Bookman, 2006.

GOODRICH, M. T.; TAMASSIA, R. **Estruturas de dados e algoritmos em Java**. Porto Alegre: Bookman, 2007.

ORACLE. **Class string**. 2010. Disponível em: <http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html>, acesso em: 24.fev.2014.

PUGA, S.; RISSETI, G. **Lógica de programação e estruturas de dados: com aplicações em Java**. São Paulo: Prentice Hall, 2008.

RAMOS, Fabio Pestana. **Introdução à lógica aristotélica**. Disponível em: <http://fabiopestanaramos.blogspot.com.br/2011/10/introducao-logica-aristotelica.html>, acesso em: 2.fev.2014.

ZIVIANI, Nívio. **Projeto de algoritmos com implementações em Java e C++**. São Paulo: Cengage Learning, 2006.