



Curso de Java

Estruturas básicas
Pacotes/Bibliotecas
Listas

Atributos e Métodos de Classe

Ministrado por: Mário Sergio

email: mariosergio30@gmail.com

<https://www.linkedin.com/profile/view?id=111098029>

Estrutura de um Programa JAVA

```
package meupacote;
```

Package. Utilizado quando o código do programa deverá fazer parte de um pacote.

```
import java.lang.*;
```

Import. Seção de importação de bibliotecas.

```
/** Nosso primeiro programa Java  
    Conhecendo a estrutura de um  
    programa Java */
```

Comentários. Com sintaxe “// ... para comentários simples ou “/* */” e a mais recente “/** .. */” que permite geração de documentação automática (ferramenta javadoc)

```
public class MinhaClassePublica {  
    ....._  
    ....._  
    /** Comentário sobre o método */  
    public (private/protected) tipoRet  
    nomeMetodo(<parametros>) {  
        // código do método  
    } // fim da definição do método  
} // fim da classe
```

Método main(). Indica que a classe Java é um aplicativo que será interpretado pela máquina virtual.

Classes.

Declaração de classes, atributos e métodos do programa Java. A declaração e a definição dos métodos ocorre obrigatoriamente dentro do limite de declaração da classe.

Programando JAVA

Elementos da Orientação a Objetos no Programa

```
// Nosso primeiro programa Java
// Conhecendo a estrutura de um programa Java
public class MeuPrimeiroPrograma {
    public static void main(String arg
        System.out.println("Olá Aluno de
    main
    MeuPrimeiroPrograma
```

Classe. Como qualquer programa JAVA, esse programa exige uma classe (palavra reservada “class”). O fato de ser pública (palavra “public”) garante visibilidade em qualquer contexto de sua utilização_

Método. A impressão da mensagem “Olá Aluno de Java” se deu pela execução do método “println” da classe “System”._

Objeto. Para imprimirmos a mensagem de saída de nosso programa precisamos de um objeto “out” da classe “System” da biblioteca padrão java.lang_

Biblioteca. A organização das classes JAVA se dá na forma de bibliotecas. Nesse programa utilizamos a biblioteca padrão da linguagem JAVA (biblioteca java.lang)_

Explicando o primeiro programa JAVA

```
public class MeuPrimeiroPrograma { ..... }
```

- Classes são tipos de dados declarados com a palavra reservada **class**.
- Cada arquivo .java deve ter somente uma classe pública e essa deve ter o mesmo nome do arquivo

```
public static void main (String args[]) {...}
```

- O método “main()” deve estar entre os métodos da classe pública e será sempre por onde o aplicativo se inicia.
- O qualificador **static** indica que este é um **método de classe**. (mas não se preocupem com isso nesse momento).
- Os argumentos passados ao método “main()” são uma lista de objetos da classe String, separados por espaços em branco.

Explicando o primeiro programa JAVA

`System.out.println` (“parâmetros”) & `System.out.print`(....)

- `System.out` é o objeto de saída padrão em Java
- Permite exibir strings e outros tipos de informações na Janela de Comando (console do sistema operacional)
- `System.out.println()` exibe seus parâmetros e pula uma linha

Comentários

// Comentários:

Use comentários para esclarecer conceitos utilizados no programa. Utilize:

`//` para comentários de linha única

`/* */` para comentários de várias linhas

`/***/` em ambos os casos e quando desejar incluir o texto na documentação de seu programa (javadoc).

Sempre inicie seus programas com comentário descrevendo o propósito do mesmo.

Palavras Reservadas em Java

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

- Como toda linguagem, Java possui identificadores reservados para comandos que permitem a execução de suas instruções
- **IMPORTANTE:** você não pode utilizar palavras-chave Java como nome de variáveis, atributos, métodos ou classes.

Declaração de Variáveis

```
String primeiroNumero;
```

```
...
```

```
int numero1;
```

- A declaração de variáveis em Java segue a sintaxe

tipo *nomeVariavel*; ou

tipo *nomeVariavel1, nomeVariavel2, ...;*

- **tipo** pode ser um dos tipos da linguagem Java ou uma classe definida por seu programa Java.
- Os tipos primitivos em Java incluem inteiros (**short**, **int** e **long**), números reais (**float** ou **double**), caracteres (**char**), tipo lógico (**boolean**) e variável binária (**byte**)

Arrays

Declarando Arrays

- Arrays são objetos que ocupam espaços contíguos de memória. O programador deve especificar o tipo, nome do array e utilizar o operador **new** para reservar o espaço necessário.

```
int[] c;      // declaração do array  
c = new int[12]; // declaração e reserva de espaço do do array
```

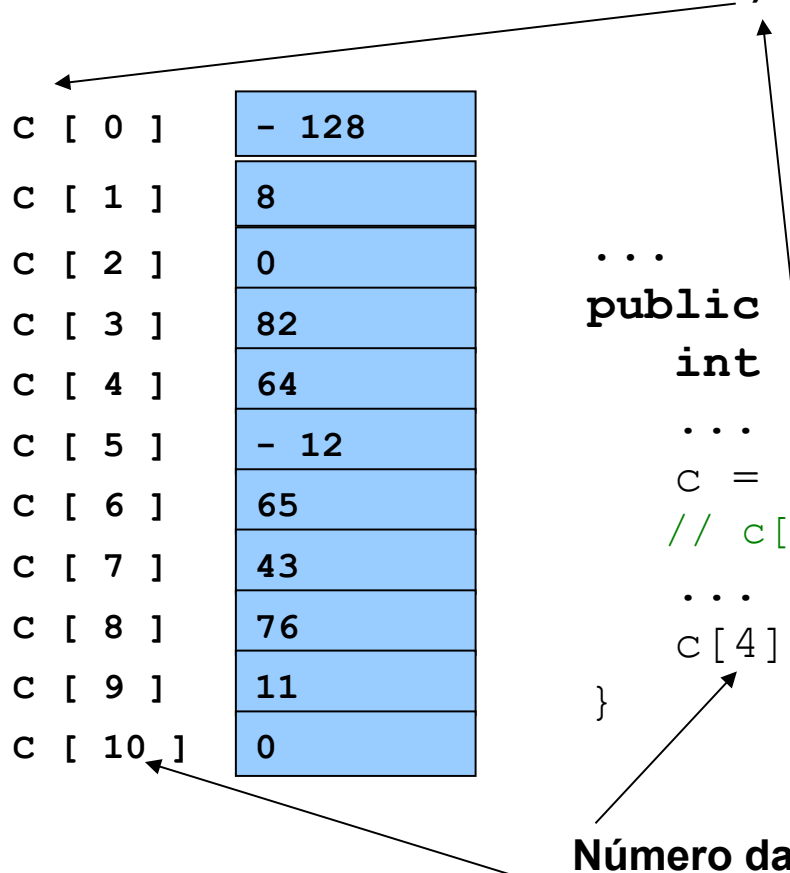
- Arrays podem ser declarados e inicializados ao mesmo tempo:

```
int[] c = {1,2,3,4,5,6,7,8,9,10,11,12};  
String[] nomes = {"lucia","maria","jose","ricardo"};
```

- Nesse caso, a reserva de espaço feita pelo operador **new** é automaticamente realizada pela máquina virtual Java.
- Quando os arrays são declarados sem inicialização, o Java faz a inicialização para zeros (variáveis numéricas), false (variáveis lógicas do tipo **boolean**) ou **null** para referências a tipos de objetos.

Arrays

Nome do *array* (todos os elementos do vetor passam a ter o mesmo nome: 'c')



c [0]	- 128
c [1]	8
c [2]	0
c [3]	82
c [4]	64
c [5]	- 12
c [6]	65
c [7]	43
c [8]	76
c [9]	11
c [10]	0

```
...  
public static void main (String args[]) {  
    int c = new int[11];  
    ...  
    c = {-128, 8, 0, 82, 64, -12, 65, 43, 76, 11};  
    // c[11] é zero por default (inicialização)  
    ...  
    c[4] += c[2];    // c[4] = 64 + 0 = 64  
}
```

Número da posição do elemento dentro de um *array* (índice ou subscrito)

Arrays

Nunca esquecer que

- Arrays em Java (como em C e C++) iniciam pela posição zero. Portanto, um array `c` de três elementos tem as posições `c[0]`, `c[1]` e `c[2]`.
- Para se encontrar o elemento de um array se usa o nome do array, seguido do subscrito (i.e., posição desejada), entre colchetes.
- Arrays em Java podem ter seu comprimento sempre conhecido pela variável `length`. Para determinar o comprimento, basta usar o nome do array, seguido de ponto e dessa variável.

Operadores de Atribuição

- **Operador de Atribuição:**

=

Não confundir com
operador de igualdade
que é **==**

- **Atribuindo valores**
exemplos:

```
nome = "joão";  
idade = 25;  
altura = 1.85;  
peso = 60.5f;  
ativo = true;
```

- **Declaração e Atribuição (Inicialização)**
em única linha:

```
String nome = "joão";  
int idade = 25;  
double altura = 1.85;  
float peso = 60.5f;  
boolean ativo = true;
```

Constantes em Java,
são identificadas com
o **qualificador final**

De devem estar em
maiúsculo

```
final double PI = 3.14;  
final String ONG = "Rede Cidadã";
```

Operadores Aritméticos

Operação de Java	Operador aritmético	Expressão algébrica	Expressão em Java
Adição	+	$f+7$	f + 7
Subtração	-	$p - c$	p - c
Multiplicação	*	bm	b * m
Divisão	/	x/y ou $x\div y$	x / y
Módulo	%	$r \bmod s$	r % s

- Java fornece, similar a outras linguagens, operadores aritméticos de adição, subtração, multiplicação, divisão e módulo.

Operadores Aritméticos de Atribuição

Operador de atribuição	Exemplo	Exemplificação	Atribui
+=	c += 7	c = c + 7	10 a c
-=	d -= 4	d = d - 4	1 a d
*=	e *= 5	e = e * 5	20 a e
/=	f /= 3	f = f / 3	2 a f
%=	g %=9	g = g % 9	3 a g

Exemplos: `int c = 3, d = 5, e = 4, f = 6, g = 12`

- Java fornece vários operadores que abreviam as expressões de atribuição;
- A simplificação de sintaxe não é a única vantagem desses operadores. Eles aumentam a velocidade de acesso às variáveis em programas.

Operadores de Igualdade e Relacionais

Operador algébrico de igualdade padrão ou operador relacional	Operador de igualdade ou relacional em Java	Exemplo de condição em Java	Significado da condição em Java
<i>Operadores de igualdade</i>			
=	==	x == y	x é igual a y
<>	!=	x != y	x não é igual a y
<i>Operadores relacionais</i>			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
=	>=	x >= y	x é maior que ou igual a y
=	<=	x <= y	x é menor que ou igual a y

- Todos os operadores relacionais têm o mesmo nível de precedência e associam da esquerda para a direita;
- Os operadores de igualdade têm o mesmo nível de precedência, que é mais baixo que a precedência dos operadores relacionais.

Operadores de Incremento e Decremento

Operador	Chamado de	Expressão	Explicação
++	pré-incremento	++a	Incrementa 'a' por 1, depois utiliza o novo valor de 'a' na expressão em que 'a' reside.
++	pós-incremento	a++	Utiliza o valor atual de 'a' na expressão em que 'a' reside, depois incrementa 'a' por 1
--	pré-decremento	--b	Decrementa 'b' por 1, depois utiliza o novo valor de 'b' na expressão em que 'b' reside.
--	pós-decremento	b--	Utiliza o valor atual de 'b' na expressão em que 'b' reside, depois decrementa 'b' por 1

- Java possui operadores que acrescentam ou diminuem valores unitários em variáveis. Seu funcionamento é semelhante à aplicação do operador de atribuição com valor 1 (ou seja: `x += 1;`), mas permite que o programador determine o momento em que deseja incrementar (ou decrementar) sua variável, com relação ao uso na expressão em que essa variável está inserida.

Operadores de Lógicos

Operador	Ação
&&	And (E)
	Or (Ou)
!	Not (Não)

Exemplo

Resultado

true & false
(8==8)&(5<3)

False
False

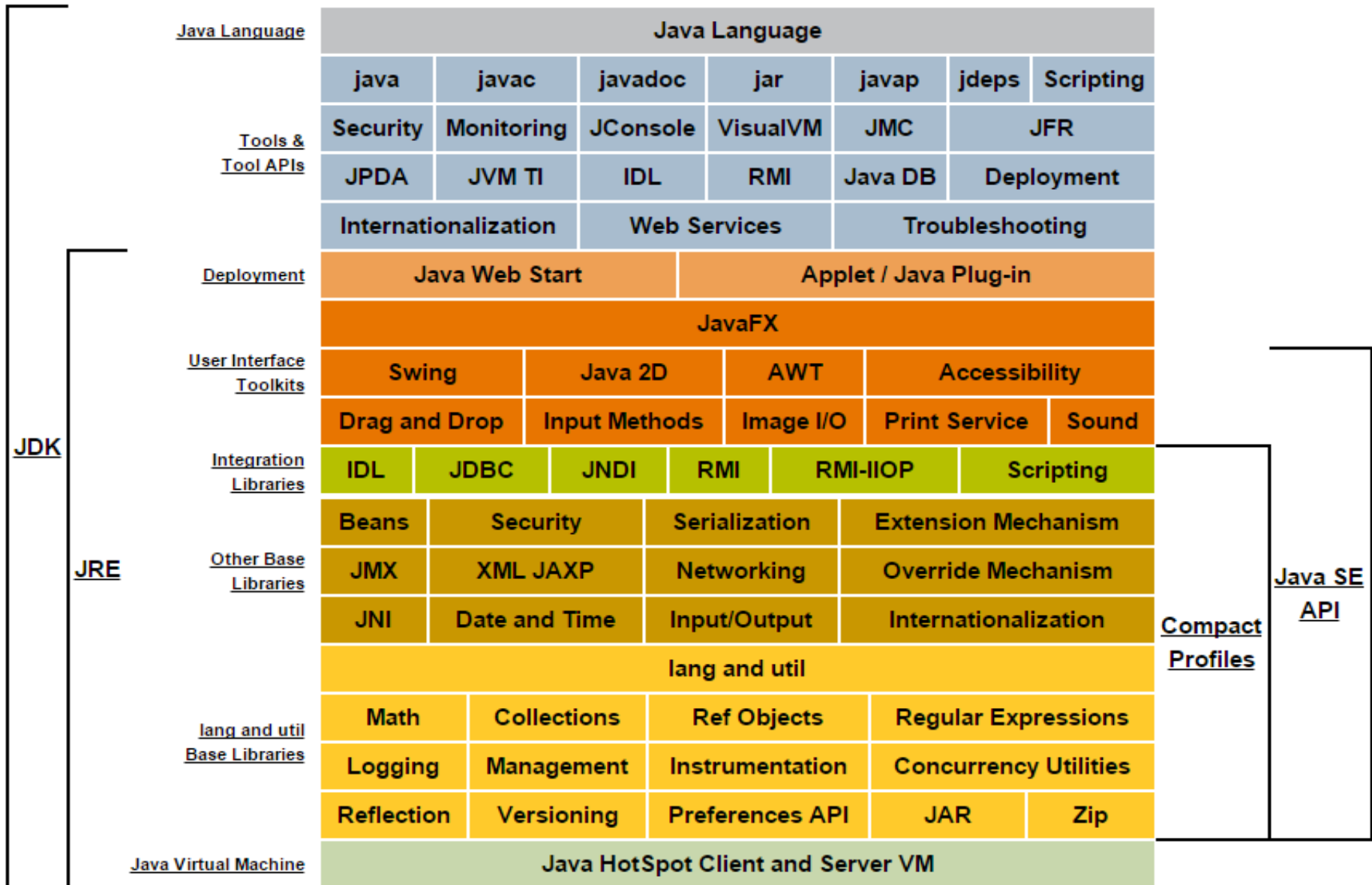
true | false
(1==1)|(9<5)

True
true

!false
!(9==9)

true
false

A Linguagem JAVA e suas Bibliotecas



Nomenclatura de Pacotes

- Além dos pacotes da API JAVA, outros pacotes podem ser desenvolvidos por terceiros. O padrão da sun para dar nome aos pacotes é relativo ao nome da empresa que desenvolveu a classe:

Exemplo de pacotes criados por terceiros:

br.com.nomedaempresa.nomedoprojeto.subpacote

br.com.nomedaempresa.nomedoprojeto.subpacote2

br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3

- Os pacotes só devem possuir **letras minúsculas**, não importa quantas palavras estejam contidas nele. Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes.
- Para criar o seu próprio pacote para sua classe, basta usar a instruções **package**

Exemplo:

package escola;

Pacotes de Classes predefinidas do JAVA

- As classes predefinidas da linguagem Java são agrupadas em categorias de classes chamadas *pacotes* (**package**), conhecidos como *bibliotecas de classes Java* ou *interface de programação de aplicativos Java* (**Java API**)
- Os nomes dos pacotes Java começam com **Java** (pacotes do núcleo da linguagem) ou **Javax** (extensões ao núcleo)

A instrução **import** é utilizada para identificar e carregar classes que desejamos utilizar em nossos programas.

- As instruções **import** devem aparecer sempre antes da definição das classes.

Exemplo:

```
import java.util.Scanner;
```

O que há na Biblioteca Java?

- A Biblioteca (*API – Application Programming Interface*) é formada por conjunto de classes do JDK, organizadas em **pacotes**;
- Exemplos de pacotes Java:
 - **java.lang**: Tipos e funcionalidades básicas da linguagem. Inclui, entre outras, as classes *String*, *Math*, *Integer* e *Thread*. É importada automaticamente em seus programas Java;
 - **java.awt**: componentes gráficos originais da linguagem (*Abstract Window Toolkit*);
 - **javax.swing**: pacote de eXtensão aos componentes gráficos com melhoramentos à biblioteca AWT
 - **java.applet**: classes específicas para tratamento de *applets*;
 - **java.net**: recursos de rede (*sockets* e URLs);
 - **java.io**: classes para escrita e leitura em arquivos;
 - **java.util**: classes para tarefas gerais, tais como vetores e *string* de *tokens*.

Utilizando Métodos

O que são Métodos

- Os métodos permitem **realizar diversas tarefas**: cálculos, comunicação com outros objetos, eles definem o comportamento de um programa.
- O métodos (**processam**) atuam sobre os dados (**variáveis/atributos**) de uma classe ou de um objeto, eles alteram o estado atual do programa.
- Em Java, *métodos* são similares aos **procedimentos e funções**. Ou seja, podem apenas **executar uma tarefa**, ou também retornar um valor para a trecho do programa que invocou (chamou o método).

Utilizando Métodos

Módulos: Pacotes de Classes+Métodos

- A melhor forma de se construir programas está na modularização, ou seja, na divisão do sistema em módulos específicos.
- Em Java os módulos são descritos por métodos e classes, que podem ser:
 - Pré-empacotados: disponíveis na Java API (bibliotecas de classes Java).
 - Métodos definidos pelo programador. Tarefas específicas que podem ser utilizadas muitas vezes por um programa.

Utilizando Métodos

Métodos de Classe ou Estáticos

-Um método é acionado (invocado) por uma *chamada de método*. Essa chamada pode ser realizada por objetos da classe, ou por chamadas diretas à classe (**no caso dos métodos estáticos**)

-Isso significa que não é necessário instanciar um objeto (operador new) para utilizá-lo. Podendo chamá-lo da seguinte maneira: **NomeDaClasse.metodo();**

-Por exemplo o método `readLine` da classe `System` é estático, é por isso o utilizamos assim: **`System.console().readLine()`**

classe

- O mesmo não acontece com os métodos `nextLine()`, `nextInt()` ... da classe `Scanner`, é por isso o utilizamos assim:

`Scanner entrada = new Scanner(System.in);`

`entrada.nextLine();`

Objeto, em breve aprenderemos mais sobre o operador new

Usando o pacote java.lang

- A classe **System** possui os métodos **out.print()**, **out.println()**, que fazem o papel do comando de saída mais básico do java.

- Também já conhecemos o método **console().readLine()** utilizado para entrada de dados a partir do console cmd.

Ex: String nome = System.console().readLine();

- A classe **String** também faz parte do pacote padrão java.lang, ela é uma classe especial que representa o tipo Character.

- Mas além da classe String, há outras classes que representam os tipos primitivos em java que são chamadas de *classes envólucro*, elas são úteis para fazer *conversão de tipos*.

Usando o pacote java.lang

As Classe envólucro (Wrappers) : *encapsulam tipos primitivos*

<i>Boolean</i>	<i>Byte</i>	<i>Character</i>
<i>Short</i>	<i>Integer</i>	<i>Long</i>
<i>Double</i>	<i>Float</i>	

Métodos de conversão a partir de uma String

Para	Método	Exemplo de expressão
int	<i>parseInt</i>	<i>Integer.parseInt(s)</i>
long	<i>parseLong</i>	<i>Long.parseLong(s)</i>
float	<i>parseFloat</i>	<i>Float.parseFloat(s)</i>
double	<i>parseDouble</i>	<i>Double.parseDouble(s)</i>
boolean	<i>valueOf,</i> <i>boolean Value</i>	<i>Boolean.valueOf(s).booleanValue()</i>

Ex: Double numero = Double.parseDouble("13.5");

Todas essas Classes possuem o método **toString()** para realizar a operação inversa.

Usando o pacote java.util

- Também já conhecemos a classe **Scanner**, também utilizado para entrada de dados do teclado.
Ex: Scanner entrada = new Scanner(System.*in*);
- A classe **Date** disponibiliza o acesso a data e hora do sistema, e pode ser utilizada em conjunto com a classe **SimpleDateFormat** do pacote **java.text**

Ex:

```
Date d = new Date();  
System.out.println(d);  
SimpleDateFormat simple = new SimpleDateFormat("dd/M/yyyy hh:mm:ss");  
System.out.println(simple.format(d));
```

Usando o pacote java.util

- A classe **Random**, é útil quando se deseja trabalhar com números aleatórios (randomicos)

Ex: sorteio de um número entre 0 e 100:

```
Random aleatorio = new Random();  
int sorteio = aleatorio.nextInt(100);  
System.out.println("Sorteio: " + sorteio);
```

Usando o pacote java.util (classe List/ArrayList)

- A classe **List** pode ser entendida como uma **coleção de objetos**, sendo mais flexível que os arrays pois podem **variar de tamanho** conforme a necessidade e também armazenam valores de **tipos diferentes**.

Para criar uma lista de nomes (String), podemos fazer:

```
List<String> listaEstados = new ArrayList<>();  
listaEstados.add("São Paulo");  
listaEstados.add("Rio de Janeiro");  
listaEstados.add("Minas Gerais");
```

```
System.out.println("Segundo elemento da lista :" + lista.get(1));
```

Usando o pacote `java.util` (classe `List/ArrayList`)

Métodos da classe `List`

add – Adiciona um item no final da coleção;

get – retorna (pega) um item em determinada posição

size – retorna o tamanho atual da coleção.

clone – Duplica a `ArrayList`

contains – busca um valor no array, e retorna `true`, se o elemento estiver no array;

indexOf – busca um valor no array, mas retorna o índice do elemento encontrado;

lastIndexOf – o mesmo que `indexOf` mas retorna o último elemento encontrado;

remove – Remove um item da coleção.

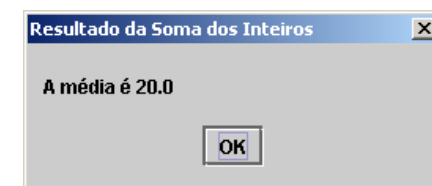
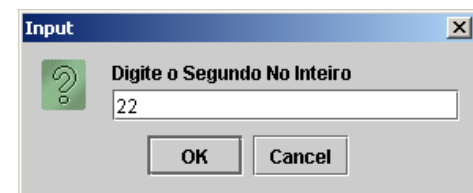
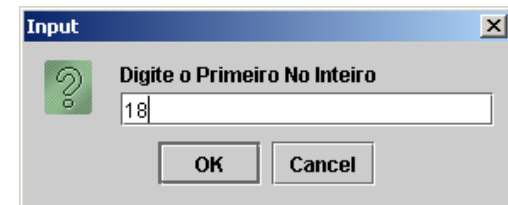
Usando o pacote javax.swing

O Pacote swing possui uma biblioteca de Classes para **interface gráfica com o usuário GUI**, por enquanto utilizaremos apenas a classe ***JOptionPane*** que oferece caixas de diálogo gráficas predefinidas que permitem aos programas exibir mensagens aos usuários:

- O método ***showMessageDialog()*** *exibe um aviso com interface gráfica para o usuário.*
- O método ***showConfirmDialog*** *exibe uma pergunta ao usuário com botões OK, NÃO E CANCELAR, e retorna um código de resposta clicada pelo usuário.*
- O método ***showInputDialog()*** *combina a montagem da janela de edição com o prompt de digitação do string fornecido pelo usuário.*
- As variáveis informadas pelo ***showInputDialog()*** *aos programas Java são sempre Strings e devem ser convertidas com o uso das classes **envólucro** do pacote java.lang.*

Exemplo de uso da Classe JOptionPane

```
// Meu Segundo Programa JAVA
// Trabalhando com Números e Operadores Aritméticos
// Baseado em Deitel & Deitel, 2003
// Pacote de extensão Java
import javax.swing.JOptionPane; // import class JOptionPane
public class Adicao {
    public static void main( String args[] )    {
        String primeiroNumero;// 1o string informado pelo usuário
        String segundoNumero; // 2o string informado pelo usuário
        int numero1;           // primeiro operando da adição
        int numero2;           // segundo operando da adição
        int media;             // Resultado da Adição
        // ler o primeiro número (na forma string)
        primeiroNumero = JOptionPane.showInputDialog("Digite o Primeiro No Inteiro" );
        // ler o segundo número (na forma string)
        segundoNumero = JOptionPane.showInputDialog( "Digite o Segundo No Inteiro" );
        // convertendo os strings em números inteiros
        numero1 = Integer.parseInt(primeiroNumero);
        numero2 = Integer.parseInt(segundoNumero);
        // Somando os números
        media = (numero1 + numero2)/2;
        // Apresentando os resultados
        JOptionPane.showMessageDialog(null, "A media é "+media,"Resultado da media: ",
            JOptionPane.PLAIN_MESSAGE);
        System.exit( 0 ); // termina a aplicação
    } // fim do método main()
} // fim da classe Adicao
```



A Classe `Math` (também do pacote `java.lang`)

Expressões Matemáticas

- Os métodos da classe `Math` permitem realizar cálculos comuns necessários em expressões matemáticas.
- Exemplos de chamadas de métodos da classe `Math`:
 - Função raiz quadrada: `double y = Math.sqrt(10.0);`
 - Função mínimo: `double z = Math.min(x, 10);`
- Os métodos da classe `Math` são **métodos estáticos**, ou seja, não necessitam de objetos da classe para sua chamada. Por essa razão você deve precer as chamadas dos métodos com o nome da classe seguido de ponto (como já fizemos nos programas anteriores):
 - `JOptionPane.showMessageDialog(...)`

Métodos da Classe Math

Método	Descrição	Exemplo
<code>abs(x)</code>	valor absoluto de x (tem versões para float , int e log)	<code>abs(23.7)</code> é 23.7; <code>abs(0.0)</code> é 0.0; <code>abs(-23.7)</code> é 23.7
<code>ceil(x)</code>	arredonda o valor de x para o menor inteiro não menor que x	<code>ceil(9.2)</code> é 10.0; <code>ceil(-9.8)</code> é -9.0;
<code>cos(x)</code>	co-seno trigonométrico de x (x em radianos)	<code>cos(0.0)</code> é 1.0
<code>exp(x)</code>	método esponencial e^x	<code>exp(1.0)</code> é 2.718281828
<code>floor(x)</code>	arredonda o valor de x para o maior inteiro não menor que x	<code>floor(9.2)</code> é 9.0; <code>floor(-9.8)</code> é -10.0
<code>log(x)</code>	logaritmo natural de x (base e)	<code>log(2.718282)</code> é 1.0; <code>log(7.389056)</code> é 2.0
<code>max(x, y)</code>	maior valor entre x e y (também em versões para float , int e long)	<code>max(2.3, 12.7)</code> é 12.7; <code>max(-2.3;-12.7)</code> é -2.3
<code>min(x, y)</code>	menor valor entre x e y (também em versões para float , int e long)	<code>min(2.3, 12.7)</code> é 2.3; <code>min(-2.3;-12.7)</code> é -12.7
<code>pow(x, y)</code>	x elevado à potência y (x^y)	<code>pow(2.0, 7.0)</code> é 128.0; <code>pow(9.0,0.5)</code> é 3.0
<code>sin(x)</code>	seno trigonométrico de x (x em radianos)	<code>sin(0.0)</code> é 0.0
<code>sqrt(x)</code>	raiz quadrada de x	<code>sqrt(900.0)</code> é 30.0; <code>sqrt(9.0)</code> é 3.0
<code>tan(x)</code>	tangente trigonométrica de x (x em radianos)	<code>tan(0.0)</code> é 0.0

Métodos da Classe Math

Chamada de Métodos

- Métodos podem ser chamados em declarações de variáveis ou como parâmetros de outros métodos

```
float z = sqrt(4.0*x);
```

```
System.out.println(Math.sqrt(x+y*f));
```

Constantes

A classe Math possui duas constantes importantes em programas matemáticos:

```
Math.E = 2.7282818284590452354  
// valor base de logaritmos naturais
```

```
Math.PI = 3.14159265358979323846  
// relação entre a circunferência e o diâmetro de círculos
```

Estrutura de Seleção `if`

A estrutura `if`

- Necessária sempre que os programas encontrarem seqüências alternativas de ações, dependendo do valor de determinada condição.
- Exemplo:

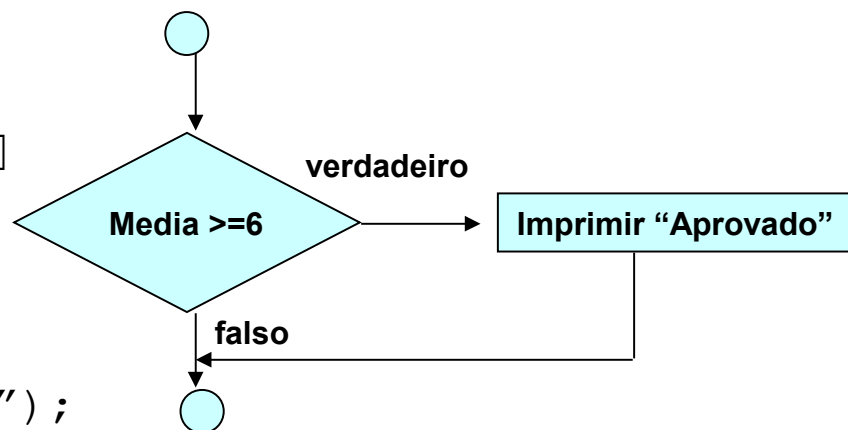
*Se a média das notas do aluno for maior ou igual a 6
Imprimir “Aprovado”*

Sintaxe

```
if (condição)  
    comando Java;  
[ou {bloco de comandos Java;}]
```

O Exemplo em Java

```
if (media >= 6)  
    System.out.println("Aprovado");  
if (media >= 6) {  
    System.out.print("O Aluno está");  
    System.out.println("Aprovado");  
} // fim do bloco if
```



Estrutura de Seleção `if/else`

A estrutura `if/else`

- Necessária sempre o programa deve executar uma ou mais ações quando uma condição for verdadeira ou, quando essa for falsa, executar outra ação ou seqüência de ações.
- Exemplo:
 - Se a média das notas do aluno for maior ou igual a 6*
Imprimir “Aprovado”
 - Senão*
Imprimir “Reprovado”

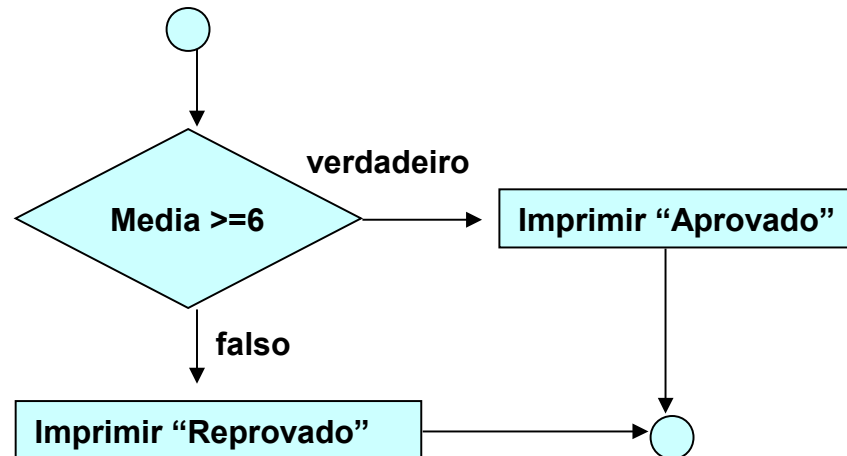
Sintaxe

```
if (condição)
    comando Java;
    [ou {bloco de comandos Java;}]
else
    comando Java;
    [ou {bloco de comandos Java;}]
```

Estrutura de Seleção if/else

O Exemplo em Java

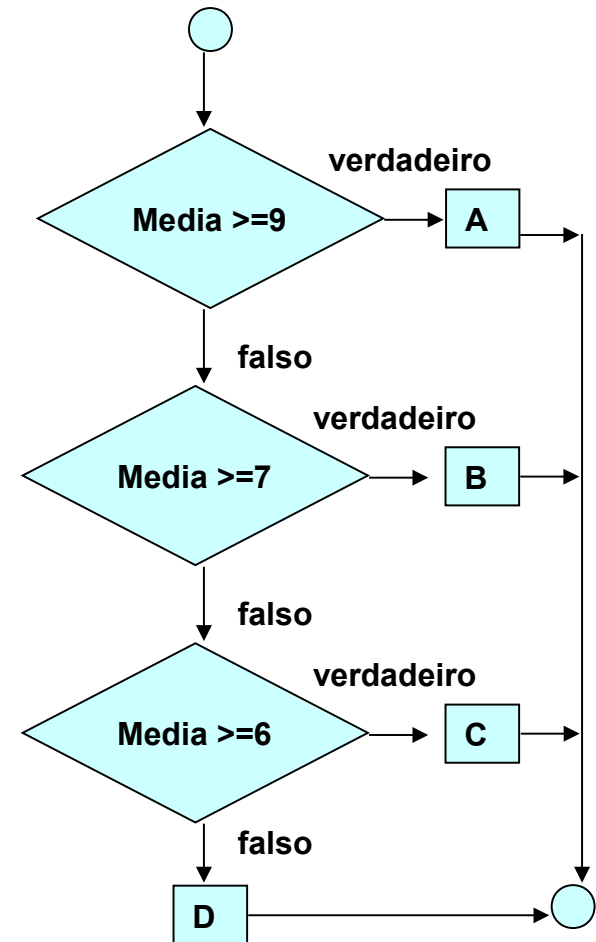
```
if (media >= 6) {  
    System.out.print("O Aluno está");  
    System.out.println("Aprovado");  
} // fim do bloco if  
  
else {  
    System.out.print("O Aluno está");  
    System.out.println("Reprovado");  
} // fim do bloco else
```



Estruturas if/else Aninhadas

Em Java

```
if (media >= 9)
    System.out.print("O Conceito é A");
else
    if (media >= 7)
        System.out.print("O Conceito é B");
    else
        if (media >= 6)
            System.out.print("O Conceito é C");
        else
            System.out.print("O Conceito é D");
```



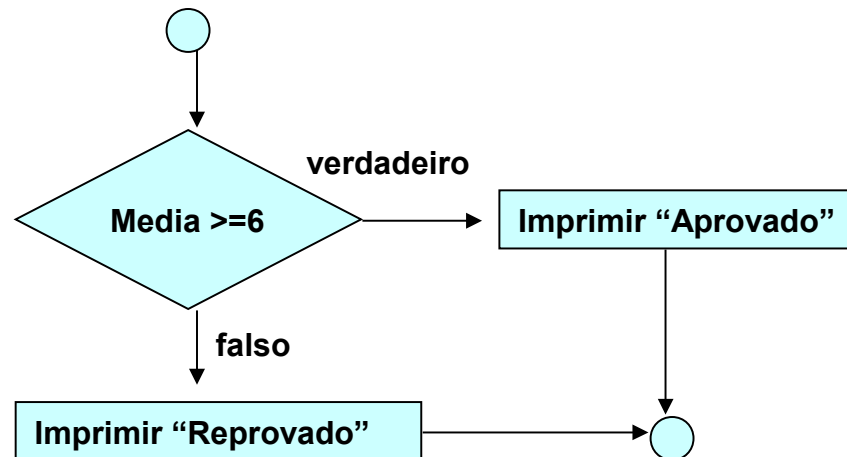
Operador Ternário Condicional ? :

Sintaxe Também conhecido como if imediato

(condição) ? {ação ou bloco verdade} : {ação ou bloco falso}

O Exemplo em Java

```
System.out.println(media >= 6 ? "Aprovado" : "Reprovado");
```



Estrutura de Seleção Múltipla switch

switch/case

- Utilizada em programas em que uma variável ou expressão pode assumir diferentes valores e há uma ação (ou bloco de ações) para cada valor possível.
- Exemplo (organizando as ações de um programa):

De acordo com a opção solicitada pelo usuário:

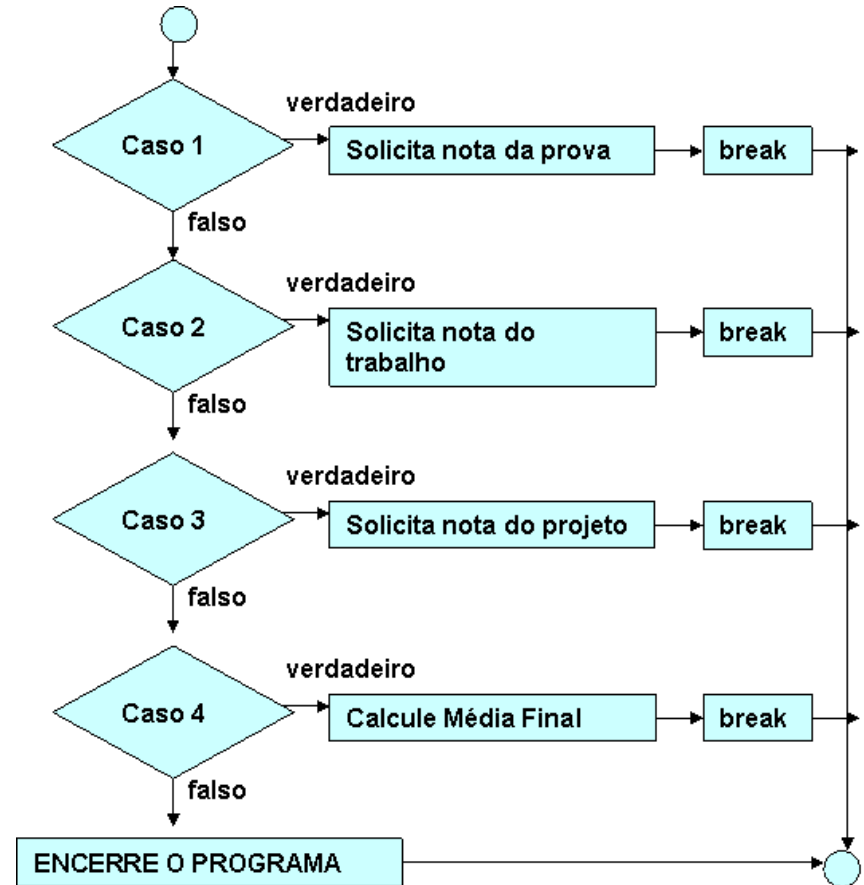
caso 1: solicite a nota da prova do aluno

caso 2: solicite a nota do trabalho do aluno

caso 3: solicite a nota do projeto do aluno

caso 4: calcule a média final do aluno

default: encerre o programa



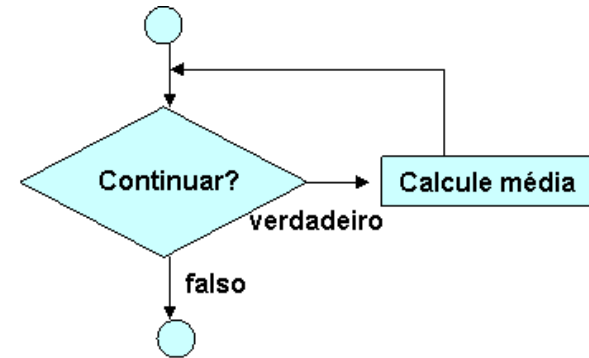
Utilize o comando **break** para não acionar as ações nos blocos definidos nos “cases” (e no default) abaixo do “case” acionado.

Estrutura switch/case

```
// programa exemplo de utilização da estrutura switch/case
import javax.swing.JOptionPane;
public class SwitchCase {
    public static void main(String arg[]) {
        int notaProva = 0, notaTrab = 0, notaProj = 0;
        float mediaFinal = 0;
        String esc;
        int escolha;
        esc = JOptionPane.showInputDialog ("Digite sua Escolha : ");
        escolha = Integer.parseInt(esc);
        switch (escolha) {
            case 1: notaProva= Integer.parseInt(JOptionPane.showInputDialog("Nota da Prova: "));
            case 2: notaTrab = Integer.parseInt(JOptionPane.showInputDialog("Nota do Trabalho: "));
            case 3: notaProj = Integer.parseInt(JOptionPane.showInputDialog("Nota do Projeto: "));
            default: if(escolha<4) mediaFinal = (notaProva + notaTrab + notaProj)/(3-escolha+1);
        }
        JOptionPane.showMessageDialog(null,"Media Final: "+
            mediaFinal,"Resultados",JOptionPane.INFORMATION_MESSAGE);
        System.exit( 0 );
    } // fim do main
} // fim da classe pública
```

Estrutura de Repetição **while**

- A estrutura de repetição **while** permite especificar uma ação ou um bloco de ações que devem permanecer sendo repetidas enquanto determinada condição for verdadeira.
- *Exemplo:*
 - *Enquanto o usuário desejar continuar calcule a média de cada aluno*
- O corpo da estrutura **while** pode ser uma instrução única ou um bloco de comandos.
- Quando a condição do comando **while** se tornar falsa, a ação (ou bloco) do comando será pulada. O programa continuará com a ação imediatamente após o comando **while**.
- **IMPORTANTE:** você deve sempre prever o comando ou ação que tornará falsa a condição do comando **while**. Caso contrário seu programa entrará em loop.

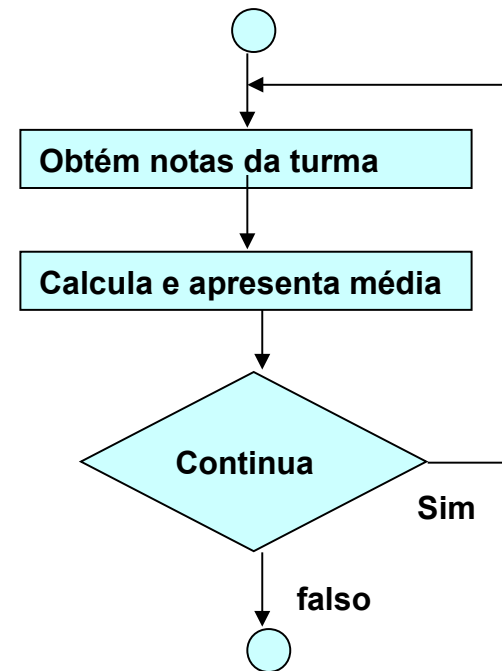


Estrutura while

```
// programa exemplo de utilização da estrutura de repetição while
import javax.swing.JOptionPane;
public class SwitchCase {
    public static void main(String arg[]) {
        int notaProva = 0, notaTrab = 0, notaProj = 0;
        float mediaFinal = 0;
        String esc;
        int escolha = 1;
        while ((escolha >=1) && (escolha <=4)){
            esc = JOptionPane.showInputDialog ("Digite sua Escolha : ");
            escolha = Integer.parseInt(esc);
            switch (escolha) {
                case 1: notaProva= Integer.parseInt(JOptionPane.showInputDialog("Nota da Prova: "));
                case 2: notaTrab = Integer.parseInt(JOptionPane.showInputDialog("Nota do Trabalho: "));
                case 3: notaProj = Integer.parseInt(JOptionPane.showInputDialog("Nota do Projeto: "));
                default: if(escolha<4) mediaFinal = (notaProva + notaTrab + notaProj)/(3-escolha+1);
            }
            JOptionPane.showMessageDialog(null,"Media Final: "+
                mediaFinal,"Resultados",JOptionPane.INFORMATION_MESSAGE);
            mediaFinal = notaProva = notaTrab = notaProj = 0;
        } // fim do while
    }
    System.exit( 0 );
} // fim do main
} // fim da classe pública
```

Estrutura de Repetição do...while

- A estrutura de repetição **do/while** permite repetir uma ação ou um bloco de ações até que determinada condição seja verdadeira. A diferença para a estrutura **while** está no fato de que **do/while** inicia pela execução do bloco e somente após a mesma analisa a condição.
- *Exemplo:*
 - Faça*
 - Obtenha as notas da turma*
 - Calcula e Apresente a média*
 - Enquanto houver mais turmas para calcular a média*
- O **do/while** sempre admite que a primeira interação ocorre antes da confirmação da condição
- O corpo da estrutura **do/while** pode ser uma instrução única ou um bloco de comandos.
- Quando a condição do comando **do/while** se tornar falsa, o programa continuará com a ação imediatamente após o comando **do/while**.
- **IMPORTANTE:** você deve sempre prever o comando ou ação que tornará falsa a condição do comando **do/while**. Caso contrário seu programa entrará em loop.



Estrutura de Repetição for

- A estrutura de repetição **for** permite repetir uma ação ou um bloco de ações com controle de contador ou da condição de permanência no looping.

- *Exemplo:*

*Para o contador 'i' de 1 a 10 faça
Obtenha a nota do i-ésimo aluno
some a nota do i-ésimo aluno ao total;
Calcule a média da turma como sendo o total dividido por 10*

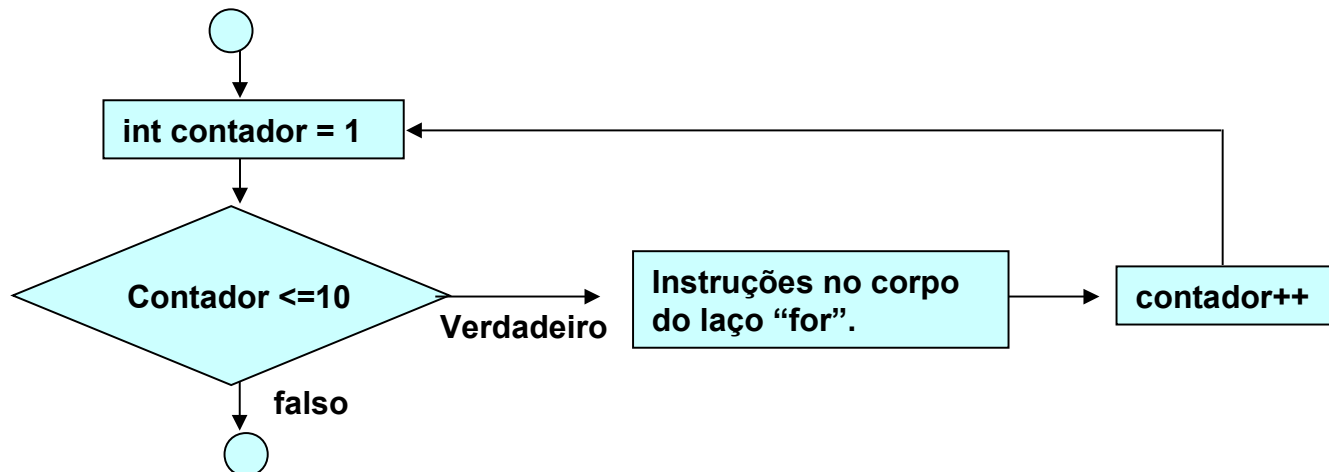
- O comando **for** tem três seções de variáveis:

for (*inicializadores; condição de continuação; incrementos*)

{

ação ou bloco de ações no comando;

}



Instruções **break** e **continue**

- As instruções **break** e **continue** modificam o comportamento das estruturas de repetição **while**, **for**, **do/while** ou **switch**. .
- A instrução **break** interrompe o laço (no caso das estruturas de repetição) e impede a execução de outros casos de um comando **switch**.
- *Exemplo:*
Enquanto verdade permanente
(ex: x == 1, sem mudar x)
realize as operações
*se condição de fim for alcançada **break**;*
Fim do Enquanto
- A instrução **continue** permite o salto do conjunto de operações, com retorno à expressão condicional do laço, reiniciando o mesmo (portanto, ao contrário do **break**, não interrompe o laço).
- Normalmente **break** e **continue** interrompem laços em que estão inseridos. Para interromper um conjunto aninhado de estruturas, deve-se utilizar **break** e **continue** rotulados (com denominação de blocos).

Exercícios

*Uma turma de alunos se submeteu a um **provão**. As notas da prova são valores inteiros no intervalo de 0 (zero) a 100 (cem).*

Prepare um programa que solicite ao usuário as notas de vários alunos (apenas uma por aluno) calcule a média da turma (isto é: some todas as notas e divida pelo total de alunos que fizeram a prova).

Após cada nota informada, deve-se perguntar ao usuário se deseja incluir mais uma nota.

No final deve-se exibir a média geral e o resultado, conforme a seguir:

Otimo (maior que 80)

Bom (Entre 60 e 80)

Regular/Ruim (menor que 60)

OBS: Usar interface gráfica para entrada e saída de dados

Declaração de Atributos

- Até agora utilizamos em nossos exemplos apenas **variáveis locais**, dentro do método main.
- Se desejarmos que nossas variáveis sejam acessadas em outras partes do nosso programa e não apenas dentro do método main, precisamos declara-las como **ATRIBUTOS**.

```
public class Jogo {
```

```
    String jogador;
```

```
    int pontuacao;
```

Declarar fora do método main,
ou de qualquer outro método

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

```
        . . .
```

```
    }
```

```
}
```

Criando seus próprios Métodos

- Além dos métodos da API Java, o programador pode desenvolver os métodos para suas próprias classes.
- Por exemplo: suponha uma classe *ApresentaQuadrados* que tem por objetivo apresentar o quadrado da série de números pares entre o número zero e um número digitado pelo usuário.
- Pode-se criar a classe em Java por:

```
class public ApresentaQuadrados {  
    public static void main (String args[]) {  
        int numeroEntrada;  
        String respostaFinal = "";  
        // ler número digitado pelo usuário  
        // realizar laço de zero até o número digitado  
        // se o número na sequência do laço for par,  
        // chamar o método de calcular o valor ao quadrado  
        // guardar o valor resultante no String de resposta  
        // se não for, continue no laço  
        // apresentar a resposta final  
    }  
    static double calculaQuadrado (int x) {  
        return Math.pow(x,2);  
    } // fim do método calculaQuadrados da classe ApresentaQuadrados  
} // fim da classe ApresentaQuadrados
```

Declaração de Métodos

- Até agora utilizamos em nossos exemplos apenas o método main.
- Se desejarmos separar as instruções do nosso programa em partes menores precisamos criar nossos próprios **MÉTODOS**.

```
public class Jogo {  
    public static void main(String args[]) {  
        . . .  
    }
```

Declaração

Declaração de Método criado pelo usuário

```
void exibeCabecalho() {  
    System.out.println("<< Star Wars Game >>");  
}
```

Implementação
entre { }

```
}
```

Declaração de Métodos

- Assim como os atributos, os métodos devem possuir um **tipo** e um **identificador** (nome). Além disso, os métodos precisam também:
- Os parenteses (), que podem **declaração de parametros**.
- Uma **implementação** entre chaves { } (exceto para metodos abstratos, veremos depois).

```
void imprime(String mensagem) {  
    System.out.println("Aviso: " + mensagem);  
}
```

Tipo de retorno nulo,
ou seja não retorna valor,
é similar a um procedimento

```
double soma(double n1, double n2) {  
    return n1 + n2;  
}
```

Parametros que devem
ser passados na chamada
ao método.

```
void somaPontos(int pontoGanho) {  
    pontuacao = pontuacao + pontoGanho;  
}
```

Tipo de retorno double,
ou seja retorna valor inteiro,
é similar a uma função

O qualificador Static

- O qualificador **static** deve ser utilizado no início da declaração de atributos e métodos quando se deseja que eles sejam acessados diretamente dentro do método main, ou em qualquer outra parte do programa sem a necessidade de criar um objeto para isso.

```
public class Jogo {  
  
    static String jogador;  
    static int pontuacao;  
  
    static void imprime(String mensagem) {  
        System.out.println("Aviso: " + mensagem);  
    }  
  
    static double soma(double n1, double n2) {  
        return n1 + n2;  
    }  
  
    static void somaPontos(int pontoGanho) {  
        pontuacao = pontuacao + pontoGanho;  
    }  
}
```

Dessa forma, temos
Atributos e Métodos de
Classe

*Obs: por enquanto, ainda não
queremos usar o operador new para
criar objetos, por isso precisamos
declarar como estáticos.*

Chamada de Métodos

- Em qualquer parte da **classe Aula1** poderemos acessar os seus atributos de classe, e chamar (invocar) seus os métodos, conforme a seguir:

```
public static void main(String args[]) {  
  
    jogador = "Michael";  
    pontuacao = 0;  
  
    imprime("Somando dois números");  
    double resultado = soma(10,20);  
}
```

- Para **acessá-los a partir de outra classe**, é necessário incluir o nome da classe que se deseja acessar na frente do identificador do atributo ou variável, separado por um ponto:

```
Aula1.jogador = "Michael";  
Aula1.pontuacao = 0;  
Aula1.imprime("Somando dois números");  
double resultado = Aula1.soma(10,20);
```

Um exemplo mais interessante

```
public class Jogo {  
  
    static String jogador;  
    static int pontuacao;  
  
    public static void main(String args[]) {  
        jogador = "Michael";  
        pontuacao = 0;  
  
        exibeCabecalho();  
        imprime(jogador + " Boa Sorte !");  
        somaPontos(20);  
        imprime("Pontuação: " + pontuacao);  
        somaPontos(80);  
        imprime("Pontuação: " + pontuacao);  
        somaPontos(50);  
        imprime("Pontuação Final: " + pontuacao);  
    }  
  
    static void exibeCabecalho() {  
        System.out.println("<< Star Wars Game >>");  
    }  
  
    static void imprime(String mensagem) {  
        System.out.println("Aviso: " + mensagem);  
    }  
  
    static void somaPontos(int pontoGanho) {  
        pontuacao = pontuacao + pontoGanho;  
    }  
}
```

*Obs: em breve conheceremos
as diretivas de encapsulamento:
public, private e protected.*

*Quando essa diretiva é
omitida o Java assume que
a visibilidade dos atributos
e métodos é pública.*

Exercícios

- Com base no exercício **provão** da pag 48:

1-Crie métodos (quantos forem necessários) para realizar as tarefas de entrada de dados, cálculo e saída de dados.

2-Guardar as notas, e agora também os nomes dos alunos, em dois objetos do tipo List do pacote java.util. Mostrar relatório com nomes e notas de cada aluno.

Referências

Programação de computadores
em Java

Rui Rossi dos Santos

Java 8: Programação de
Computadores - Guia Prático de
Introdução, Orientação e
Desenvolvimento - José Augusto
N. G. Manzano

Slides do Prof. Roberto Pacheco
INE – CTC – UFSC

Sugestão de Material

<http://www.dialetodigital.com/blog/conteudos-programacao/>