



Instituto de Informática

Programação Orientada a Objetos

**Revisão da Programação Estruturada em
uma Linguagem Orientada a Objetos
(Java)**

28/09/2023

Prof. Dirson S. Campos



Tipos Primitivos em Java

- ❑ São oito tipos básicos:
 - boolean, char
 - byte, short, int, long
 - float, double
- ❑ Possuem mesmo tamanho e características, independentemente da plataforma do Sistema Operacional.

Tipos Primitivos em Java

| Tipo | Tamanho em bits | Valores | Padrão |
|---|-----------------|--|--------------------------------------|
| boolean | | true or false | |
| [Nota: A representação boolean é específica da Java Virtual Machine em cada plataforma] | | | |
| char | 16 | '\u0000' a '\uFFFF' (0 a 65535) | (Conjunto de caracteres Unicode ISO) |
| byte | 8 | -128 a +127 (-2^7 a $2^7 - 1$) | |
| short | 16 | - 32768 a +32767 (-2^{15} to $2^{15} - 1$) | |
| int | 32 | - 2147483648 a +2147483647 (-2^{31} a $2^{31} - 1$) | |
| long | 64 | - 9223372036854775808 a + 9223372036854775807 ($- 2^{63}$ a $2^{63} - 1$) | |
| float | 32 | Intervalo negativo: - 3.4028234663852886E+38 a - 1.40129846432481707E-45 Intervalo positivo: 1.40129846432481707E-45 a 3.4028234663852886E+38 | (ponto flutuante IEEE 754) |
| double | 64 | Intervalo negativo: -1.7976931348623157E+308 a -4.94065645841246544E-324 Intervalo positivo: 4.94065645841246544E-324 a 1.7976931348623157E+308 | (ponto flutuante IEEE 754) |

Cuidado com os faixas de valores de um tipo primitivo, caso contrário pode-se obter resultados inesperados

```
public class tamanhoByte {  
    public static void main(String[ ] args) {  
        byte i = 127;  
        System.out.println("Valor de i = " + i);  
        i++;  
        System.out.println("Valor de i + 1 = " + i);  
        i++;  
        System.out.println("Valor de i + 2 = " + i);  
    }  
}
```

Saída:

```
Valor de i = 127  
Valor de i + 1 = -128  
Valor de i + 2 = -127
```

Observação: Os resultados esperados seria 127, 128 e 129, bem diferente dos resultados obtidos (127, -128, -127). Logo, cuidado com os faixas de valores de um tipo primitivo, caso contrário, pode-se obter resultados inesperados como acima.



Tipos Ponto Flutuante

- ❑ Entre dois números reais existem infinitos números que ocupariam um espaço infinito de memória é todos os computadores possuem memória finita. Logo os números reais são representados com precisão de casas decimais.
- ❑ O padrão de float é até 6 casa decimais e double até 15 casas decimais.
- ❑ A linguagem Java seguem o padrão científico IEEE 754. O IEEE (*Institute of Electrical and Electronics Engineers*) *pode ser acessado em* (<http://www.ieee.org/portal/site>)



Exemplo precisão de um número de ponto flutuante

```
public class PontoFlutuante {  
    public static void main(String args[]) {  
  
        double d = 1.23000087513252000092785E+2;  
  
        System.out.println("Valor de d = " + d);  
        float d1 = (float)d; //Transformando double em float  
        System.out.println("Valor de d1= " + d1);  
  
    }  
}
```

Saída:

```
Valor de d = 123.000087513252  
Valor de d1= 123.000084
```



Tipo Caractere

☐ Representação de Caracteres Individuais

☐ Tamanho de 16 bits

☐ Tabela Unicode

- código numérico sem sinal (de 0 a 65535)
- Internacionalização (<http://unicode.org/>)
- compatível com a tabela ASCII (*American Standard Code for Information Interchange*) que, em português, significa "Código Padrão Americano para o Intercâmbio de Informação") é uma codificação de caracteres de sete bits baseada no alfabeto inglês. Desenvolvida a partir de 1960, grande parte das codificações de caracteres modernas a herdaram como base, inclusive a computação.



Tipo Caractere

- ❑ Valor literal de um caractere. Caracteres em java são colocados em aspas simples. Ex. 'a'.
- ❑ Existem caracteres especiais sem representação visual. Os mais utilizados são:

Representação

\n

\r

\b

\t

\f

\'

\"

\\

Significado

Pula linha (*newline* ou *linefeed*)

Retorno de carro (*carriage return*)

Retrocesso (*backspace*)

Tabulação (*horizontal tabulation*)

Nova página (*formfeed*)

Apóstrofe

Aspas

Barra invertida



Tipo Caractere - Exemplo

```
public class charTeste {  
    public static void main(String[ ] args) {  
        char i = 'a';  
        System.out.println("i: " + i);  
        i++;  
        System.out.println("i: " + i);  
        i++;  
        System.out.println("i: " + i);  
        i = (char) (i * 1.2);  
        System.out.println("i: " + i);  
        int j = i;  
        System.out.println("j: " + j);  
    }  
}
```

Saída:

```
i: a  
i: b  
i: c  
i: v  
j: 118
```



Tipo Booleano

- ❑ Variáveis booleanas (declara **boolean** em Java) representam um tipo de dado que permite apenas dois valores lógicos a saber:
 - true (verdadeiro) ou
 - false (false)
- ❑ Não são tipo primitivo de dados em C.



Tipo Booleano - Exemplo

```
public class TestaBooleano {  
    public static void main(String args[]) {  
        boolean a = false;  
        boolean b;  
        if (a == true) {  
            System.out.println("a é verdadeiro");  
        } else { System.out.println("a é falso."); }  
        b = !a; //B recebe o valor lógico não A  
        if (b == true) { System.out.println("b é verdadeiro.");  
            } else System.out.println("b é falso.");  
    }  
}
```



Tipo Booleano

(Rodando o fonte anterior)

Saída:

```
a é falso.
```

```
b é verdadeiro.
```



Declaração de Variáveis em Java

❑ Declara-se um variável colocando o tipo primeiro, seguido pelo nome da variável.

❑ **Exemplo:**

- **byte a;**
- **int umaVariavelInteira;**
- **long umaVariavellonga;**
- **char ch;**



Declaração de Variáveis em Java

□ Regras de declaração de nomes em Java

- Um nome de variável precisa começar com uma letra e ser uma seqüência de letras ou dígitos.
- Todos o caracteres no nome de uma variável são significativos e a caixa da letra, se maiúscula ou minúscula, também é significativa.
- O comprimento do nome de variável é, essencialmente, ilimitado.
- Não se pode usar uma palavra reservada em Java para um nome de variável.



Declaração de Variáveis

- ❑ Pode ser feita em qualquer ponto do programa
- ❑ Visibilidade
 - a partir de onde foi declarada
 - visível nos blocos internos
 - blocos delimitados por chaves: { e }
 - não pode haver variáveis com mesmo nome no mesmo escopo



Atribuições e Inicializações

❑ Faz-se uma atribuição a uma variável já declarada usando o nome da variável à esquerda, um sinal de igual (=) e depois uma expressão Java que tenha um valor apropriado à direita.

❑ Exemplo:

- `int a;` //esta é uma declaração
- `a =37;` //esta é uma atribuição
- `char charSim;` // declaração de uma variável do tipo caractere
- `charSim= 's';`
- `int i = 10;` // declaração com inicialização

Observação: `'s'` entre aspas simplés é um caractere e `"s"` é uma string contendo um único caractere.



Operadores de Atribuição

□ Básico

`int x = 10;` `// inicializacao`

`x = x + 12;` `// atribuicao`

`x = y = z = 15;` `// encadeada`

□ Compostos

| 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> |
|---------------------|------------------------|



Conversões entre Tipos Numéricos

☐ Pode misturar tipos

- Se algum dos operandos for do tipo double, então o outro operando será convertido em um double
- Caso contrário, se algum dos operandos for do tipo float, o outro operando será convertido em um float
- Caso contrário, se algum dos operandos for do tipo long, o outro operando será convertido em um long
- forma análoga para os tipos inteiros: int, short e byte



Conversões entre Tipos Numéricos

- ❑ Conversões onde pode haver perda de informação devem ser feitas explicitamente através do operador de cast
- ❑ Promoção
 - Ocorre quando são feitas operações com tipos inteiros (byte, short e char) menores que int
 - Resultados das operações são do tipo int
 - Racional: maior probabilidade de ocorrência de overflow nestes tipos, uma vez que o intervalo de valores é pequeno



Conversões entre tipos Numéricos

□ A sintaxe para conversões é da pela tipo do alvo em parênteses, seguido do nome da variável.

- Exemplo: `double x = 9.997;`
- `int nx = (int) x;`
- Como resultado a variável `x` terá o valor 9.



Conversões entre tipos Numéricos

❑ Para arredondar (round) um número do ponto flutuante para o inteiro mais próximo, use o método `Math.round`.

- Exemplo: `double x = 9.997;`
- `int nx = (int) Math.round(x);`
- Como resultado a variável `x` terá agora o valor 10.

Conversões permitidas são:

`byte → short → int → long → float → double → char`
`→ int`



Comentários em Java

❑ A linguagem java tem três formas de identificar os comentários:

1. Par de //

- Usa-se esta forma para um comentário que irá até o final de uma linha.

2. Combinação /* e */

- Usa-se quando são necessários comentários mais extensos.



Comentários em Java

3. Combinação `/**` e `*/`

- Tipo de comentário que pode ser usado para gerar documentação automaticamente.
 - uso de javadoc
 - gera páginas html no padrão fornecido no JDK
 - comentários imediatamente antes do elemento
 - uso de tags html
 - formatação do texto, imagens, links, tabelas
 - tags especiais javadoc `@`
 - links para outros elementos e formatação padronizada



Operadores Artiméticos básicos

□ Os operadores aritméticos básicos:

- + adição
- - subtração
- * multiplicação
- / divisão.

□ Exponenciação

- Usa-se o método chamado pow que faz parte da classe Math de Java.lang.
- Exemplo:
- `double y = Math.pow (x,a)`
- Obs: O método pow precisa de argumentos que são do tipo double e retorna um double também.

Exemplo de Operadores Aritméticos

```
public class TestaOperadoresArithmeticos
{   public static void main ( String [] args)
    { short x = 2;          int y = 3;
      double a = 7.8;       double b = 3.4;
      System.out.println ( "x é " + x + ", y é " + y );
      System.out.println ( "x + y = " + (x + y) );
      System.out.println ( "x - y = " + (x - y) );
      System.out.println ( "x / y = " + (x / y) );
      System.out.println ( "x % y = " + ( x % y ) );
      System.out.println ( "\na é " + a + ", b é " + b );
      System.out.println ( " a / b = " + ( a / b ) );
      a = x / y;
      System.out.println ( " a = " + a );   } }
```

Exemplo de Operadores Aritméticos

Saída:

x é 2, y é 3

x + y = 5

x - y = -1

x / y = 0

x % y = 2

a é 7.8, b é 3.4

a / b = 2.2941176470588234

a = 0.0



Operadores Incremento/decremento

❑ Operadores de incrementação: ++

❑ Operadores de decrementação: --

- Exemplo:
- `int n=12;`
- `n++;`
- resultado `n=13`.
- Obs: `4++` não é um opção válida.

❑ Formas de Operadores

- pós-fixado: `n++`
- pré-fixado: `++n`



Exemplos (incremento/decremento)

□ Incremento

```
int x, y, z;  
x = 10;  
y = x++;  
z = ++x;  
x = ++z + z;
```

□ Decremento

```
int x, y, z;  
x = 10;  
y = x--;  
z = --x;  
x = z-- + z;
```



Resumo de operadores aritméticos em Java

| Operador | Significado | Exemplo |
|----------|---------------------------|------------|
| + | Adição | a + b |
| - | Subtração | a - b |
| * | Multiplicação | a * b |
| / | Divisão | a / b |
| % | Resto da divisão inteira | a % b |
| - | Sinal negativo (- unário) | -a |
| + | Sinal positivo (+ unário) | +a |
| ++ | Incremento unário | ++a ou a++ |
| -- | Decremento unário | --a ou a-- |



Operadores Relacionais

- $=$ igualdade
- \neq desigualdade (diferente)
- $<$ menor que
- $>$ maior que
- \leq menor ou igual a
- \geq maior ou igual a

Exemplo Operadores Relacionais

```
public class TestaRelacional {  
    public static void main (String[] args)  
    {  
        int a = 28;        int b = 25;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("a == b -> " + (a == b));  
        System.out.println("a != b -> " + (a != b));  
        System.out.println("a < b -> " + (a < b));  
        System.out.println("a > b -> " + (a > b));  
        System.out.println("a <= b -> " + (a <= b));  
        System.out.println("a >= b -> " + (a >= b));  
        System.out.println("a = b -> " + (a = b));  
    }  
}
```



Exemplo Operadores Relacionais

Saída:

```
a = 28
```

```
b = 25
```

```
a == b -> false
```

```
a != b -> true
```

```
a < b -> false
```

```
a > b -> true
```

```
a <= b -> false
```

```
a >= b -> true
```

```
a = b -> 25
```




Operadores Lógicos

❑ Binário (exemplo)

- $a \ \&\& \ b$:AND (E): Retorna true se a e b for true, caso contrário retorna false.
- $a \ || \ b$:OR (ou): Retorna false se a e b for false, caso contrário retorna true.
- $a \ ^ \ b$: Exclusive OR “OU exclusivo”: Retorna true se A e B for logicamente diferentes, caso contrário retorna false.

❑ Unário (exemplo)

- $!a$: NOT (Não): Retorna true se A for false, retorna false se a for true.

❑ Operadores Lógicos (bit a bit)

- $\&$ (“E” lógico)
- $|$ (“OU “ lógico)
- \sim (“Negação lógica”)

Observação: Usa-se parênteses para indicar explicitamente a ordem quer se quer que as operações sejam realizadas.



Exemplo Operadores Lógicos

```
public class TestaOperadoresLogicos {  
    public static void main (String[] args)  
    { System.out.println("True & True = " + (true & true) );  
      System.out.println("True && True = " + (true && true) );  
      System.out.println("True & False = " + (true & false) );  
      System.out.println("True && False = " + (true && false) );  
      System.out.println("True | True = " + (true | true) );  
      System.out.println("True || True = " + (true || true) );  
      System.out.println("True | False = " + (true & false) );  
      System.out.println("True || False = " + (true & false) );  
      System.out.println("Not True = " + (!true) );  
      System.out.println("Not False = " + (!false) );  
    } }
```



Exemplo Operadores Lógicos

Saída:

```
True & True = true
True && True = true
True & False = false
True && False = false
True | True = true
True || True = true
True | False = false
True || False = false
Not True = false
Not False = true
```



Avaliação de Expressões

- Precedência de Operadores

- tabela de precedências

$c = a + b * c;$

- Associatividade de Operadores

- esquerda para direita, exceto atribuição

$c = a * b * c;$

$c = a = b = 10;$

- Associatividade de Operandos

- esquerda para direita

$c = b++ + b;$

Precedência/Associatividade dos operadores

| Operadores | Associatividade | Tipo |
|------------------------------|----------------------------|----------------------------|
| ++ -- | da direita para a esquerda | unário pós-fixado |
| ++ - + - ! | da direita para a esquerda | unário pré-fixado |
| * / % | da esquerda para a direita | multiplicativo |
| + - | da esquerda para a direita | aditivo |
| < <= > >= | da esquerda para a direita | relacional |
| == != | da esquerda para a direita | igualdade |
| & | da esquerda para a direita | AND (E) bit a bit |
| ^ | da esquerda para a direita | Ou Exclusivo(exclusive OR) |
| | da esquerda para a direita | OR (OU) bit a bit |
| && | da esquerda para a direita | AND (E) |
| | da esquerda para a direita | OR (OU) |
| ? : | da direita para a esquerda | condicional |
| = += -= *= /= %= | da direita para a esquerda | atribuição |



Tipos Compostos

□ Objetos

- único tipo composto
- vetores, strings, matrizes, registros e arquivos são objetos
- alocados no monte
- variáveis primitivas possuem tipo, objetos possuem classe

□ Exemplo

```
class coordenadas_espaco {  
    public int x;  
    public int y;  
    public int z;  
}
```



Programação Estruturada

- ❑ Programação estruturada é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e repetição (iteração).
- ❑ Tendo, na prática, sido transformada na Programação Modular, a Programação Estruturada orienta os programadores para a criação de estruturas simples em seus programas, usando as subrotinas (procedimentos ou rotina) e as funções, em java são chamadas de métodos.
 - **Os algoritmos são ensinados as próximas gerações em um pseudocódigo na língua natal do estudante (português no Brasil) e em notação matemática.**



Quem definiu a Programação Estruturada ?

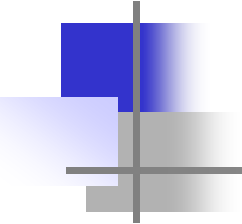
- Bohm e Jacopini¹ definiram que apenas três formas de controle são necessárias para implementar um algoritmo em programação estruturada:
 - **Seqüência** (instruções executadas em seqüência, na ordem que aparecem no código)
 - **Seleção** (estrutura condicional) e
 - **Repetição** (estrutura de repetição).

1 Bohm, C., and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," Communications of the ACM, Vol. 9, No. 5, May 1966, pp. 336-371.



Implementação da Programação Estruturada em Java

- ❑ A programação Estruturada da Linguagem Java é baseada na sintaxe da Linguagem C/C++;
- ❑ Estrutura de Seqüência é trivial, as instruções em um bloco qualquer é executada na ordem que são lidas.
- ❑ A seleção é implementada em Java de quatro maneiras básicas:
 - Instrução if (seleção única);
 - Instrução if ... Else (seleção dupla);
 - Instrução switch (seleção múltipla);
 - Operador ternário.



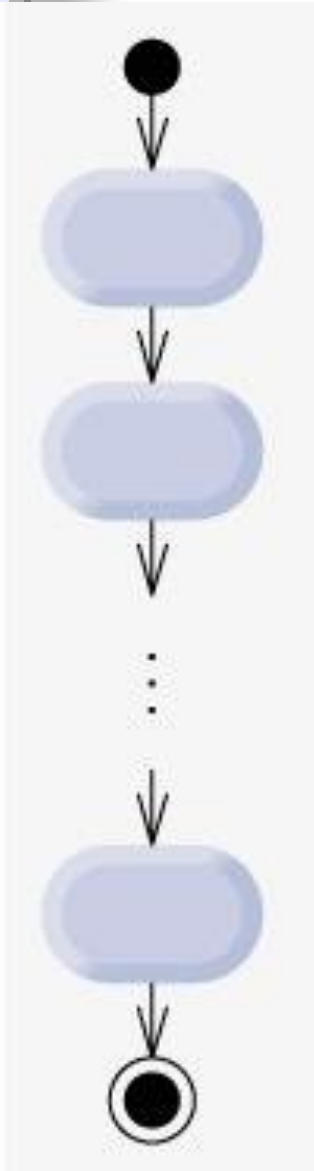
Implementação da Programação Estruturada em Java

□ A repetição é implementada em Java de três maneiras:

- Instrução while;
- Instrução do ... while;
- Instrução for.

Estrutura Seqüencial

Exemplo UML - Java



Ponto inicial da seqüência.

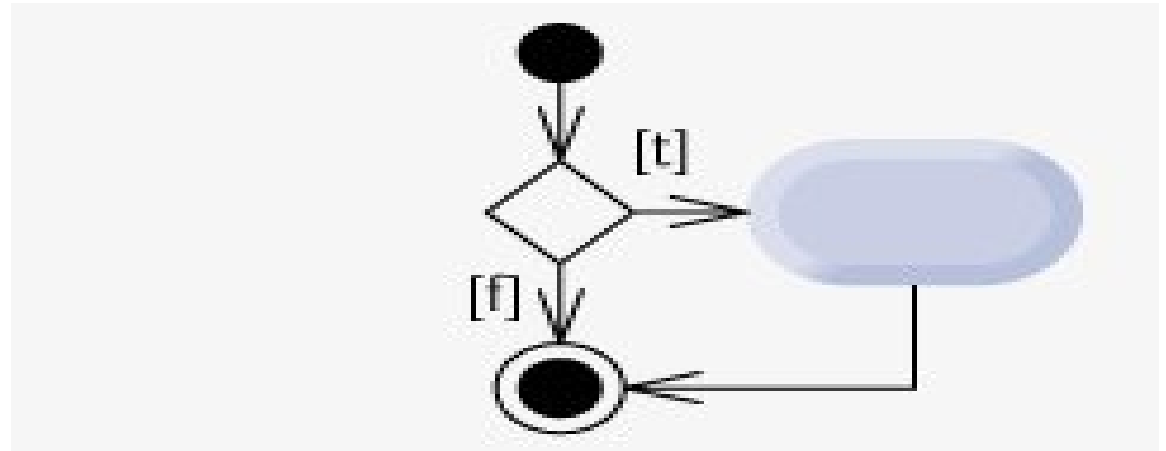
```
public class SomaVetor {
    public static void main( String[] args )
    {   int total = 0;
        int[ ] vetor = { 0, 1, 2, 3, 4, 5 };
        for ( int i=0; i < vetor.length; i++ )
            total += vetor[i];
        System.out.println( "A soma é: "
                            + total );
    }
}
```

Ponto final da seqüência.

Saída:
A soma é: 15
43

Estrutura Condicional ou de Seleção

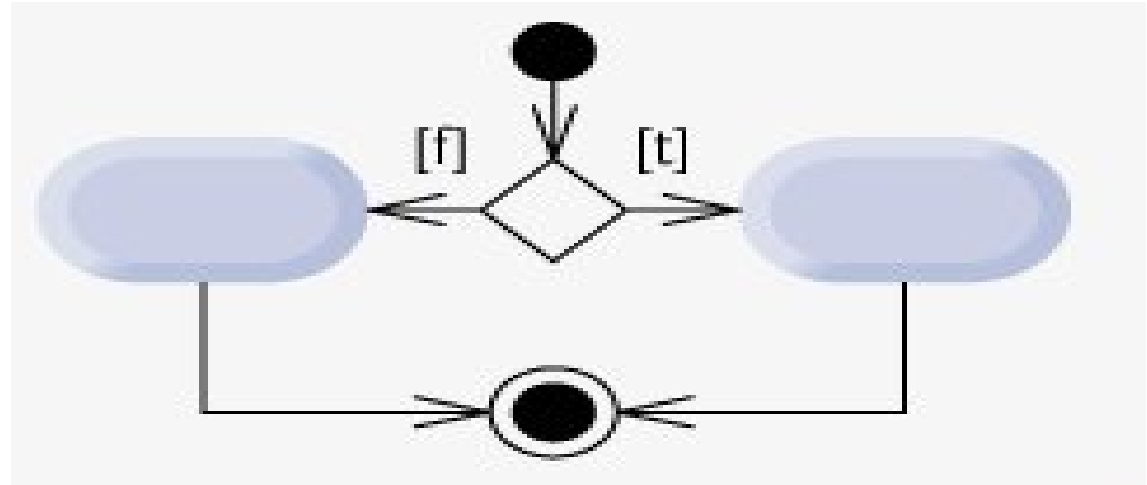
Instrução if
(seleção única)



```
if (condições)
// se condição for verdadeira faça
{
    instruções válidas em Java;
}
```

Estrutura Condicional ou de Seleção

Instrução if-else (seleção dupla)

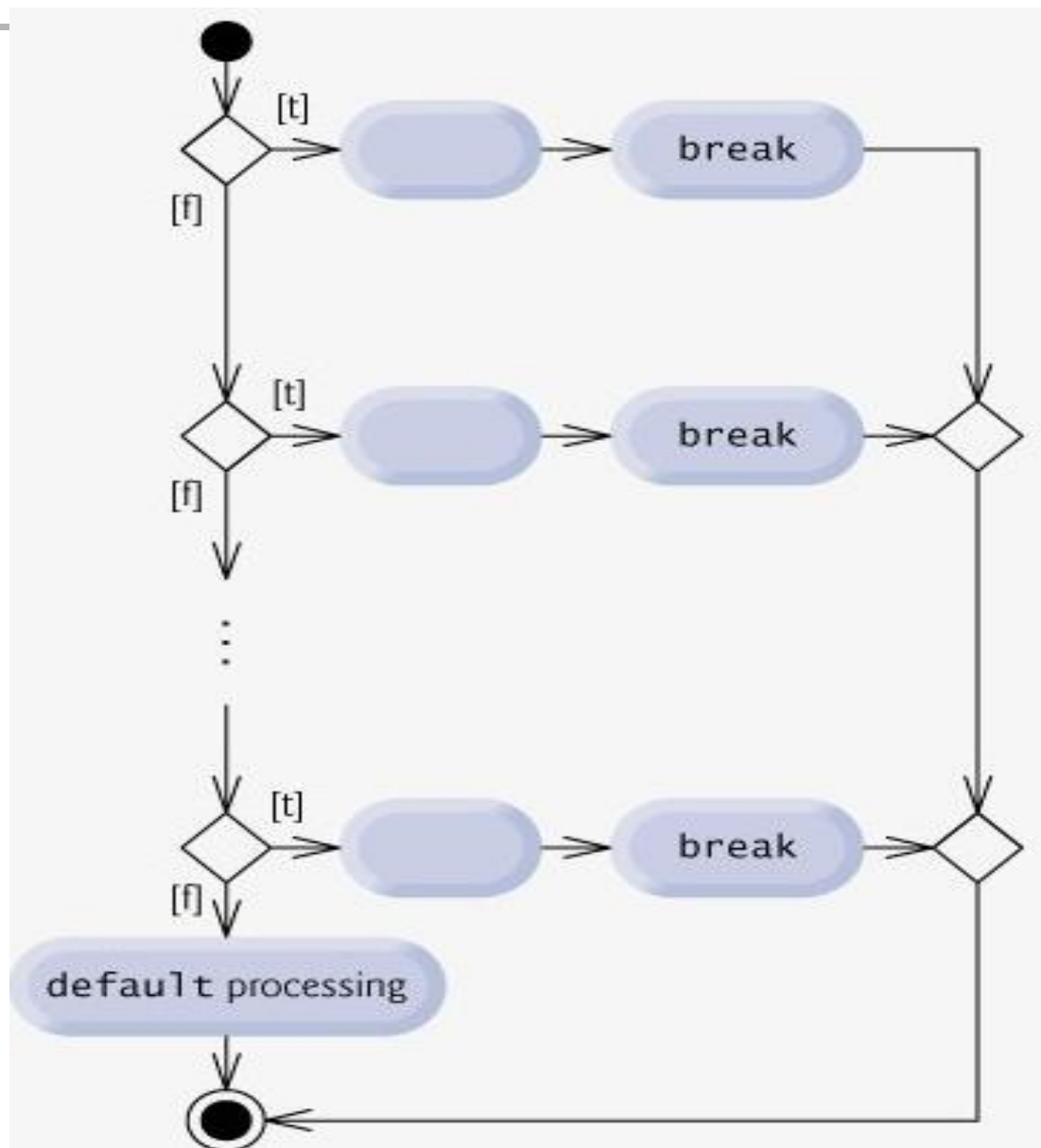


```
if (condição)
// se condição for verdadeira faça
{
instruções válidas em Java;
} else { // se condição for falsa faça
instruções válidas em Java;
}
```

Estrutura Condicional ou de Seleção

Instrução switch com breaks (seleção múltipla)

```
switch (condição) {  
  case ABC:  
    instruções ;  
    break;  
  case DEF:  
    instruções ;  
    break;  
  default:  
    instruções ;  
    break;  
}
```





Estrutura Condicional ou de Seleção por Operador Ternário

- ❑ O operador ternário é uma forma compacta de realizar um if-else a sintaxe é:

<expressão_booleana> ? <valor_1> : <valor_2>

- ❑ A sintaxe equivalente na estrutura condicional if-else é:

if (<expressão_booleana>)

{ // Se a expressão for verdadeira <valor_1>

} else

{ // Se a expressão for falsa <valor_2>

}

Estrutura Condicional ou de Seleção por Operador Ternário

```
public class TestaNario
{
```

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

```
        int i = 7;
```

```
        int j;
```

```
        j = i < 10 ? (2*i) : (i);
```

```
        // se i < 10 temos j = 2*i, caso contrário j = i
```

```
        System.out.println("i: " + i); // "logo i = 7"
```

```
        System.out.println("j: " + j); // "logo j = 14"
```

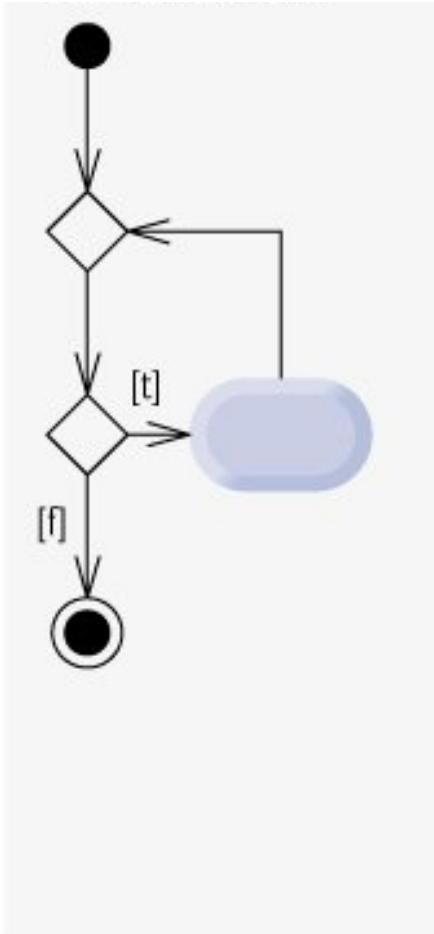
```
    } }
```

Saída:

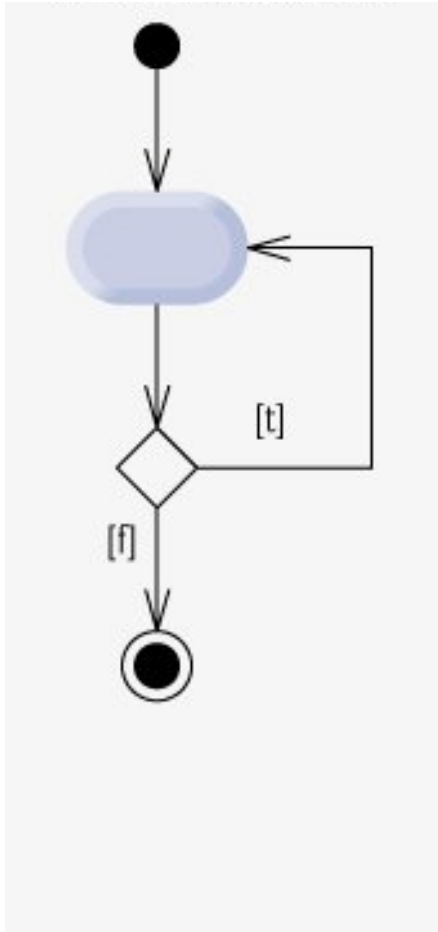
```
i: 7
j: 14
```


Estrutura de Repetição - while

```
while (condição ) {  
    // faça enquanto a condição  
    // permanecer verdadeira  
    instruções válidas em Java;  
}
```



Estrutura de Repetição do ... while



```
do { //corpo  
  instruções válidas em Java;  
} while (condição);
```

A instrução de repetição do ... while é semelhante à instrução while. Em while o programa testa a condição de execução do laço em seu início. Se a condição for falsa, o corpo nunca executa.

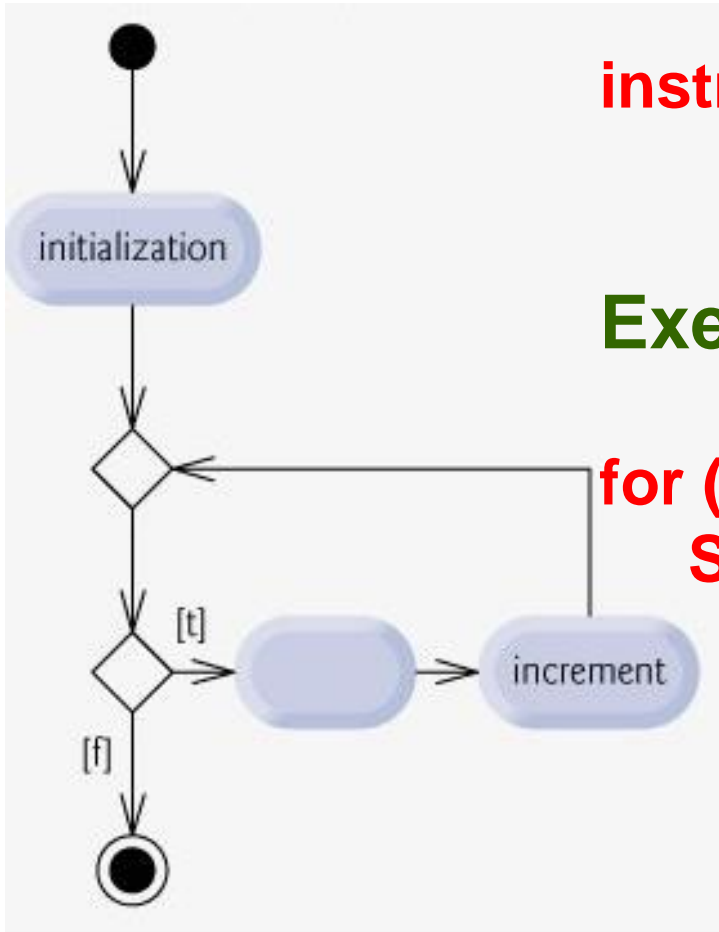
Em do ... while este teste é feito no final, isto garante que ao menos uma vez o corpo será executado.

Estrutura de Repetição for

```
for (Inicialização ; Condição de parada;  
      Alteração do contador) {  
  instruções válidas em Java;  
}
```

Exemplo numérico:

```
for (int i = 10; i > 0; i--)  
  System.out.println("Contagem  
    decrescente" + i);
```





Resumo da sintaxe de vetores (declaração e inicialização)

□ Exemplos:

```
int a[]; //declara vetor de inteiros
```

```
a=new int[3];
```

```
//cria um objeto vetor com 3 elementos
```

```
//as duas linhas anteriores podem ser abreviadas por:
```

```
int a[]=new int[3];
```

```
//pode-se inicializar o vetor na declaração
```

```
int a[3]={0,1,2};
```



Exemplo de vetores em Java

```
import java.util.*;
public class TestaVetor
{
    public static void main( String args[] ) {
        int i, n = 5;
        int vetor[ ] = new int [n];
        Scanner ler= new Scanner(System.in);
        System.out.printf( "Digite os %d elementos do vetor: \n", n);
        for( i =0; i < n; i++){
            vetor[i]=ler.nextInt();
        }
        System.out.print( "O vetor = [ ");
        for (i = 0; i < n; i++ ) {
            System.out.printf( " %d ", vetor[ i ]); }
        System.out.println( " ] ");
    }
}
```



Exemplo de vetores em Java

Saída:

```
Digite os 5 elementos do vetor:
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
0 vetor = [ 1 2 3 4 5 ]
```



Resumo da sintaxe de matrizes (declaração e inicialização)

□ Exemplos:

```
int a[][]; //declara uma matriz de inteiros
```

```
a=new int[3][3];
```

```
//cria um objeto matriz com 3 linhas e 3 colunas
```

```
//as duas linhas anteriores podem ser abreviadas por:
```

```
int a[][]=new int[3][3];
```

```
//pode-se se quiser inicializar a matriz na declaração
```

```
int a[3][3]={ {0,1,2},{3,4,5},{6,7,8} };
```



Exemplo de matriz em Java

```
public class TestaMatriz {  
    public static void main( String args[] ) {  
        int n = 2;  
        int matriz[][] = {{1,2}, {3,4}};  
        // imprimindo a matriz inicializada  
        System.out.printf( "Matriz de n = %d que foi inicializada é:\n", n);  
        for (int i = 0; i < n; i++ ) {  
            System.out.print( "[ ");  
            for (int j = 0; j < n; j++ ) {  
                System.out.printf( " %d ", matriz[i][j]);  
            }  
            System.out.println( " ] ");  
        }  
    }  
}
```




Exemplo de matriz em Java

Saída:

Matriz de $n = 2$ que foi inicializada é:

[1 2]

[3 4]