



Quem se prepara, não para.

# Algoritmos e Programação

Michelle Hanne


michelle.andrade@newtonpaiva.br

# ***Sumário:***

- ✓ Uso de valores Inteiros e Reais
- ✓ Formatação da Saída
- ✓ Laço while e do-while
- ✓ Break e Continue

```
byte   b = 127;  
short  s = 32767;  
int     i = 2147483647;  
long    l = 9223372036854775807L;
```

Valores `long` são representados com  
um `L` ou `l` no final  
(caso contrário são `int`)



Valores `float` são representados  
com um `F` ou `f` no final  
(do contrário são `double`)

```
float f = 3.4028235E38F;  
double d = 1.7976931348623157E308;
```

Estes são os  
limites positivos

$7.1E2 = 7.1 \times 10^2$   
 $7.1e2 = 7.1 \times 10^2$

```
import java.util.Scanner();
```

Representa  
o teclado



```
Scanner in = new Scanner(System.in);  
String nome    = in.nextLine();  
int    idade   = in.nextInt();  
double salario = in.nextDouble();
```


```
double x = 10000.0 / 3.0;
```

```
System.out.print(x);
```




3333,3333333333335

```
System.out.printf("%.3f", x);
```



3.333,333

```
System.out.printf("R$ %, .2f", x);
```



R\$ 3.333,33

A formatação de casas decimais irá utilizar as configurações regionais do computador

O método `printf` recebe mais de um argumento, o primeiro sempre é o formato (tipo string - indica o texto que será impresso), seguido por valores.

```
System.out.printf(formato, valor1, valor2, ...);
```

Existem vários conversores e os mais utilizados são:

<code>%d</code>	<code>int</code>
<code>%c</code>	<code>char</code>
<code>%s</code>	<code>String</code>
<code>%f</code>	<code>double e float</code>

<code>printf("%.2f", 10.5);</code>	10.50	Formata com 2 casas decimais
<code>printf("%,d", 17435);</code>	17,435	Formata separando na casa dos milhares
<code>printf("%02d", 6);</code>	06	Formata com 2 dígitos, completando com zeros
<code>printf("%+f", 13.7);</code>	+13.700000	Formata forçando a exibição do sinal

A forma geral do laço **while** é

```
while (condição) instrução;
```

onde instrução pode ser uma única instrução ou um bloco de instruções, e condição define a condição que controla o laço. A condição pode ser qualquer expressão booleana válida. O laço se repete enquanto a condição é verdadeira. Quando a condição se torna falsa, o controle do programa passa para a linha imediatamente posterior ao laço



## Laço while

```
// Demonstra o laço while.
class WhileDemo {
    public static void main(String args[]) {
        char ch;
        // exhibe o alfabeto usando um laço while
        ch = 'a';
        while(ch <= 'z') {
            System.out.print(ch);
            ch++;
        }
    }
}
```

O laço **do-while** verifica sua condição no fim do laço. Ou seja, um laço **do-while** será sempre executado pelo menos uma vez. A forma geral do laço **do-while** é:

```
do { instruções;  
} while(condição);
```

Embora as chaves não sejam necessárias quando há apenas uma instrução presente, elas são usadas com frequência para melhorar a legibilidade da estrutura do-while, evitando, assim, confusão com **while**. O laço **do-while** é executado enquanto a expressão condicional for verdadeira.

# Laço do-while

```
// Adivinhe a letra do jogo, 4a versão.
class Guess4 {
public static void main(String args[]) throws java.io.IOException {
char ch, ignore, answer = 'K';
do {
    System.out.println("I'm thinking of a letter between A and Z.");
    System.out.print("Can you guess it: ");

    // lê um caractere
    ch = (char) System.in.read();

    // descarta qualquer outro caractere do buffer de entrada
    do {
        ignore = (char) System.in.read();
    } while(ignore != '\n');

    if(ch == answer)
        System.out.println("** Right **");
    else {
        System.out.print("...Sorry, you're ");
        if(ch < answer) System.out.println("too low");
        else System.out.println("too high");
        System.out.println("Try again!\n");
    }
} while(answer != ch);
}
}
```

É possível forçar a saída imediata de um laço, ignorando o código restante em seu corpo e o teste condicional, com o uso da instrução **break**. Quando uma instrução **break** é encontrada dentro de um laço, este é encerrado e o controle do programa é retomado na instrução posterior ao laço:

```
// Usando break para sair de um laço.
class BreakDemo {
    public static void main(String args[]) { int num;
        num = 100;
        // executa o laço enquanto i ao quadrado é menor do que num
        for(int i=0; i < num; i++) {
            if(i*i >= num) break; // encerra o laço se i*i >= 100
            System.out.print(i + " ");
        }
        System.out.println("Loop complete.");
    }
}
```

É possível forçar uma iteração antecipada de um laço, ignorando sua estrutura de controle normal. Isso é feito com o uso de **continue**. A instrução **continue** força a ocorrência da próxima iteração do laço e qualquer código existente entre ela e a expressão condicional que controla o **laço é ignorado**. Logo, **continue** é basicamente o complemento de **break**:

```
// Usa continue.
class ContDemo {
    public static void main(String args[]) {
        int i;
        // exibe os números pares entre 0 e 100
        for(i = 0; i<=100; i++) {
            if((i%2) != 0) continue; // iterate
            System.out.println(i);
        }
    }
}
```

SANTOS, Marcela Gonçalves dos. **Linguagem de programação**. SAGAH, 2018. ISBN digital: 9788595024984.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Bookman, 2018. ISBN digital: 9788582604694.

SCHILDT, Herbert. Java para iniciantes. Disponível em: Minha Biblioteca, (6th edição). Grupo A, 2015.

SILVA, Fabricio Machado da. **Paradigmas de programação**. SAGAH, 2019. ISBN digital: 9788533500426.