

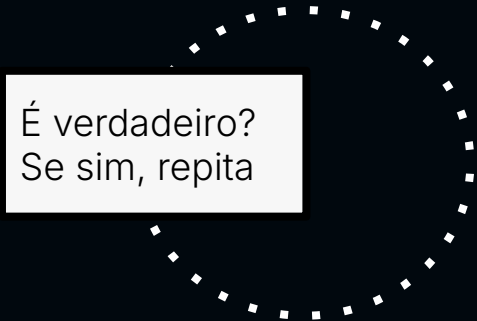
Laços de repetição

O que são laços de repetição?

Laços são estruturas que permitem **executar um bloco de código repetidamente**, enquanto uma condição for verdadeira. Estruturas de laços que temos em Java:

- For
- While
- Do...While

A leitura de um laço pode ser interpretada como: **repita até que** a condição se torne falsa.



É verdadeiro?
Se sim, repita

Sintaxe do laço FOR

O laço **for** é utilizado para executar um bloco de código para cada elemento do iterável. Ele é útil quando você **sabe exatamente quantas iterações** deseja realizar.

for (**inicialização** ; **condição** ; **atualização**)

bloco de código

Executada uma única vez antes do loop começar.

Avaliada antes de cada iteração. Se for **true**, o loop continua; se for **false**, o loop para.

Executada após cada iteração do loop, geralmente incrementando ou decrementando um valor.

Exemplo de código FOR

Esse exemplo calcula a tabuada de um número qualquer armazenado na variável **numero** e imprime na tela cada um deles, indicando o número, o multiplicador e o resultado.

```
int numero = 5;

for (int i = 1; i <= 10; i++) {
    System.out.printf("%d X %d = %d \n", i, numero, i * numero);
}
```

O termo "i" é apenas um identificador que usamos para representar cada elemento da iteração enquanto percorremos com o laço for. Você pode substituir "i" por qualquer outro nome de variável que faça sentido para o seu contexto.

Primeiro, criamos uma variável **numero** com o valor que desejamos para cálculo da tabuada

Em seguida, utilizamos um **laço for** para percorrer os valores de 1 a 10 e exibir o cálculo da tabuada na tela.

Sintaxe do laço WHILE

O laço **while** executa um bloco de código enquanto uma condição especificada for **verdadeira**. Ele é útil quando você **não sabe exatamente quantas iterações** serão necessárias, já que a execução depende do resultado da condição a cada iteração.

```
while ( condicao )  
    bloco de código
```

A condição é qualquer expressão que resulte em um valor booleano (*True* ou *False*). Exemplo: *while numero < 30*.

Operadores de comparação

Os operadores de comparação são utilizados para comparar valores, retornando **True** ou **False**, dependendo da condição estabelecida.

Operador	Conceito	Exemplo
> (Maior que)	Verifica se um valor é maior que outro	<code>x > 10</code>
< (Menor que)	Verifica se um valor é menor que outro	<code>x < 10</code>
== (Igual a)	Verifica se um valor é igual a outro	<code>x == 10</code>
!= (Diferente de)	Verifica se um valor é diferente de outro	<code>x != 10</code>
>= (Maior ou igual a)	Verifica se um valor é maior ou igual a outro	<code>x >= 10</code>
<= (Menor ou igual a)	Verifica se um valor é menor ou igual a outro	<code>x <= 10</code>

Exemplo de código WHILE

Esse exemplo faz um contador regressivo e imprime na tela a cada 1 segundo.

```
int contador = 10;
while (contador > 0) {
    System.out.printf("%d \n", contador);
    Thread.sleep(1000);
    contador -= 1;
}
```

Dentro do laço, imprimimos o valor atual do contador e, em seguida, decrementamos o contador em 1, com `contador -= 1`. Esse processo se repete até que a condição `contador > 0` se torne falsa (ou seja, quando contador chegar a 0).

Definimos uma variável contador com o valor 10. O contador serve para **garantir que a condição eventualmente se torne falsa.**

O laço while verifica a condição `contador > 0`. **Enquanto essa condição for verdadeira**, o bloco de código dentro do while será executado.

Sintaxe do laço DO...WHILE

O laço **do...while** tem uma sintaxe similar a do **while**, visto anteriormente, sendo útil quando não se sabe exatamente quantas iterações serão necessárias. A sua diferença é que ela faz com que o bloco seja executado **pelo menos uma vez**, mesmo que a condição seja **falsa**.

do {

bloco de código

} while (**condicao** **)**

A condição é qualquer expressão que resulte em um valor booleano (*True* ou *False*). Exemplo: *while numero < 30*.

Exemplo de código DO...WHILE

Esse exemplo faz um contador regressivo e imprime na tela a cada 1 segundo.

```
int contador = 0;
do {
    System.out.printf("%d \n", contador);
    Thread.sleep(1000);
    contador -= 1;
} while (contador > 0);
```

Dentro do laço, imprimimos o valor atual do contador e, em seguida, decrementamos o contador em 1, com `contador -= 1`. Esse processo se repete até que a condição `contador > 0` se torne falsa (ou seja, quando contador chegar a 0).

Definimos uma variável contador com o valor 0. O contador serve para **garantir que a condição eventualmente se torne falsa.**

O laço **do...while** faz com que o bloco de código seja executado **pelo menos uma vez**, mesmo o contador já iniciado com 0, tornando a condição **falsa**.

Loop Infinito

Um loop infinito é quando um laço de repetição continua a **executar sem parar**, porque a condição para sair dele nunca se torna falsa.

```
int contador = 10;

while (contador < 10) {
    System.out.printf("%d \n", contador);
    Thread.sleep(1000);
} ;
```


Neste exemplo, o laço continuará para sempre imprimindo o número **10**, porque a condição `contador < 10` **nunca mudará** para falsa. Isso pode ocorrer tanto em laços for quanto while.

Break

Mas e se quisermos interromper a execução de um laço antes que ele termine naturalmente? É aí que a instrução `break` entra em cena. O `break` permite que você saia imediatamente de um laço, mesmo que a condição para continuar seja **verdadeira**.

```
String[] nomes = {"Ana", "Bruno", "Carlos", "Jacqueline", "Eduardo"};

for (int i = 0; i <= 5; i++) {
    if (nomes[i].equals("Jacqueline")) {
        System.out.println("Professora do curso: " + nomes[i]);
        break;
    } else {
        System.out.println("Nome: " + nomes[i]);
    }
};
```



Neste exemplo, o laço irá percorrer os nomes e, ao encontrar "Jacqueline", imprimirá **"Professora do curso: Jacqueline"** e sairá do laço, não imprimindo os nomes que vêm a seguir.

Continue

Agora, e se quisermos pular a execução de uma iteração específica do laço, mas continuar com as demais? O continue permite que você pule para a próxima iteração do laço, ignorando o restante do código na iteração atual.

```
String[] nomes = {"Ana", "Bruno", "Carlos", "Jacqueline", "Eduardo"};

for (int i = 0; i <= 5; i++) {
    if (nomes[i].equals("Jacqueline")) {
        continue;
    } else {
        System.out.println("Nome: " + nomes[i]);
    }
} ;
```

Neste exemplo, quando o laço encontra "Jacqueline", ele vai ignorar o mesmo e sair do loop sem a imprimir o mesmo, continuando com os outros nomes na lista.

Sintaxe do laço for-each

O laço **for-each**, também conhecido como **enhanced for** (*for aprimorado*), é uma forma simplificada de percorrer arrays e coleções em Java. Ele é útil quando você precisa iterar sobre todos os elementos da lista, sem se preocupar com o índice.

```
for ( tipo variável nome variável : coleção ){  
    bloco de código  
}
```

O tipo dos
elementos do array
ou coleção.

Nome da variável
que representa
cada elemento.

O array ou coleção a ser
percorrida.

Exemplo de código for-each

Esse é o mesmo exemplo usado para exemplificar o **continue**, mas reescrito usando o **for-each loop**.

```
String[] nomes = {"Ana", "Bruno", "Carlos", "Jacqueline", "Eduardo"};

for (String nome : nomes) {
    if (nome.equals("Jacqueline")) {
        continue;
    } else {
        System.out.println("Nome: " + nome);
    }
} ;
```

A cada iteração, a variável **nome** vai representar o elemento do array **nomes**.

Como não usamos índices, o código fica mais limpo . É recomendável quando só precisamos dos valores.

Função útil em laços

- **length()** é utilizada para obter o comprimento de uma lista, string ou outro tipo de coleção. Ela nos permite saber quantas iterações precisamos realizar em um laço.

Compartilhe um resumo de seus novos
conhecimentos em suas redes sociais.

[#aprendizadoalura](#)

alura



Escola Programação