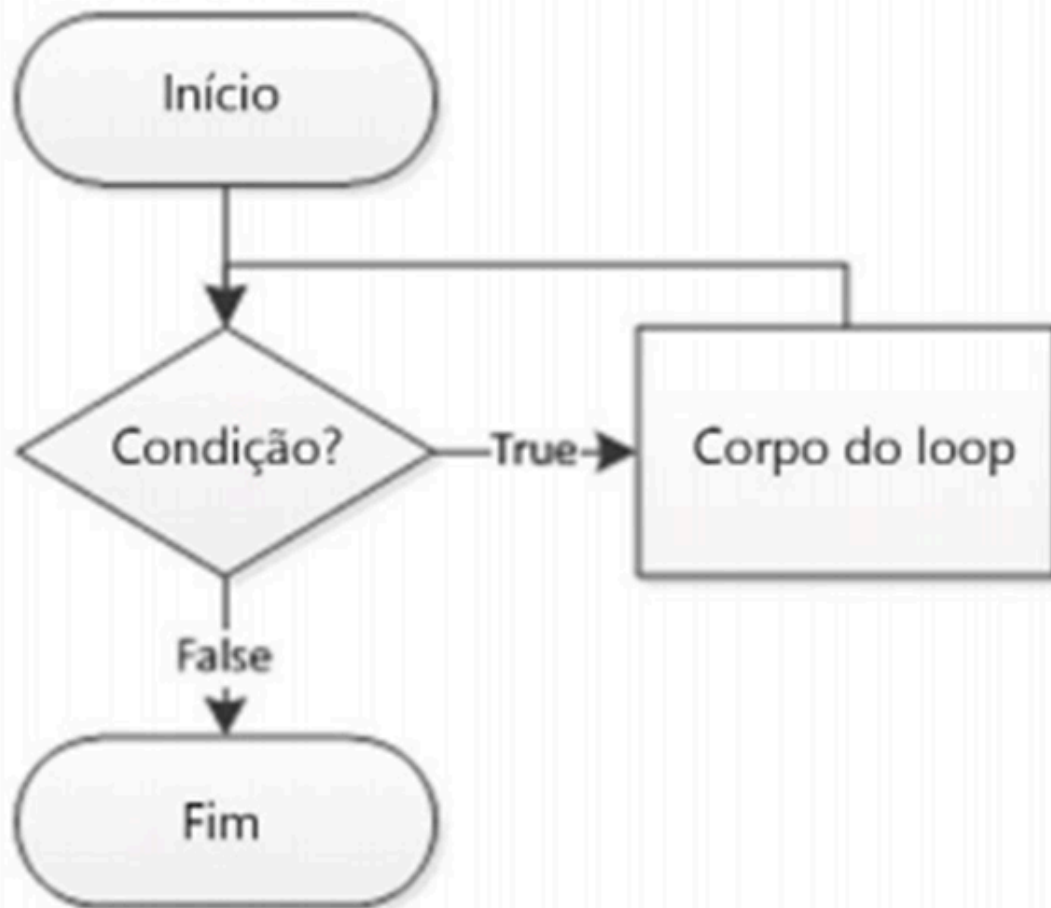




# Instruções de Loop

## Loops While em Java

Na programação de computadores, um loop é uma sequência de instruções que são repetidas continuamente até que uma determinada condição seja atingida. Imagine que você precise escrever um programa que execute uma tarefa repetitiva, como imprimir de 1 a 100. Escrever 100 declarações de impressão não seria uma boa ideia. Os loops são projetados especificamente para executar tarefas repetitivas com um conjunto de códigos. Loops economizam muito tempo. Um loop é uma estrutura que permite a execução repetida de um bloco de instruções. Dentro de uma estrutura de loop, uma expressão booleana é avaliada. Se for verdade, um bloco de instruções chamado corpo do loop é executado e a expressão booleana é avaliada novamente. Enquanto a expressão for verdadeira, as instruções no corpo do loop continuarão sendo executadas. Quando a avaliação booleana é falsa, o loop termina.



Em Java, existem três tipos de loops:

- Um loop while, no qual a expressão booleana que controla o loop é a primeira instrução do loop
- Um loop for, que geralmente é usado como um formato conciso no qual executar loops
- Um loop do ... while, no qual a expressão booleana de controle de loop é a última instrução do loop



## Loops while

O loop while é bom para cenários em que você não sabe quantas vezes um bloco ou instrução deve repetir, mas deseja continuar fazendo o loop

enquanto alguma condição for verdadeira. Uma declaração while se parece abaixo. Em Java, um loop while consiste na palavra-chave while, seguida por uma expressão booleana entre parênteses, seguida pelo corpo do loop, que pode ser uma única instrução ou um bloco de instruções cercado por chaves.

```
while (expressão) { // faz coisas }
```

Você pode usar um loop while quando precisar executar uma tarefa um número predeterminado de vezes. Um loop que executa um número específico de vezes é um loop definido ou um loop contado. Para escrever um loop definido, você inicializa uma variável de controle de loop, uma variável cujo valor determina se a execução do loop continua. Enquanto o valor booleano resultante da comparação da variável de controle do loop e outro valor for verdadeiro, o corpo do loop while continua sendo executado. No corpo do loop, você deve incluir uma instrução que altera a variável de controle do loop.

```
public class WhileLoopDemo{  
  
    public static void main(String[] args){  
  
        intvar=1;  
  
        intlimit=11;  
  
        while(var<limit)
```

```
{
```



```
System.out.println("Contador do Loop: " + var);
```

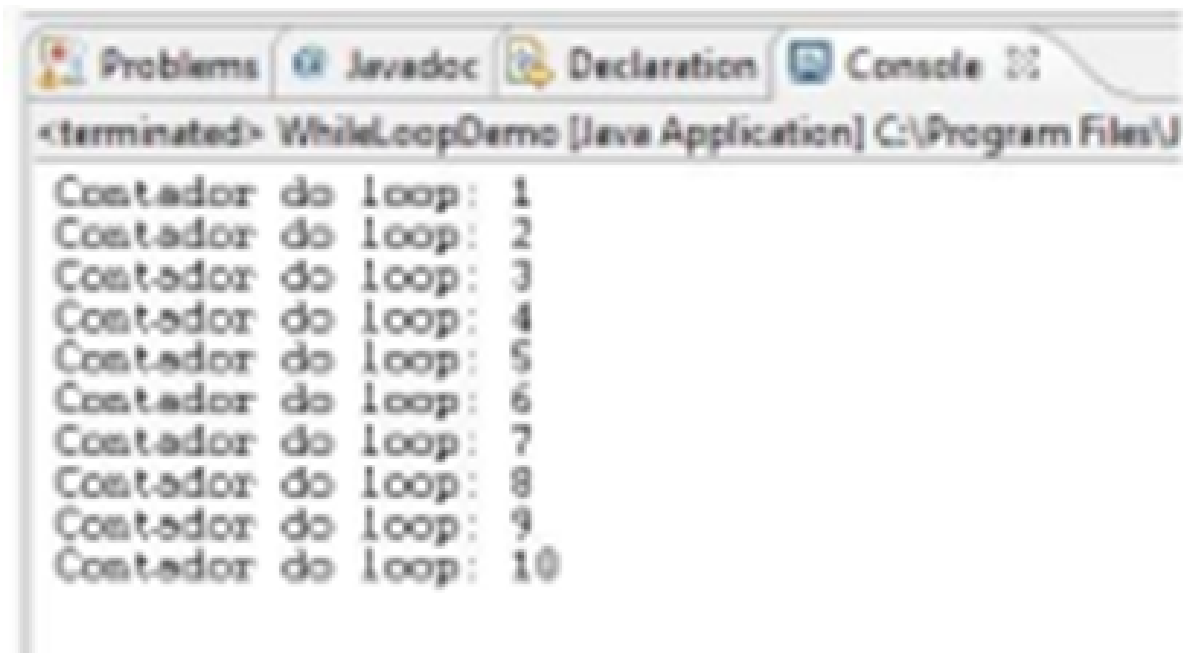
```
var++;
```

```
}
```

```
}
```

```
}
```

## Resultado:



O ponto principal a ser lembrado sobre um loop while é que ele pode nunca ser executado. Se a expressão de teste for falsa na primeira vez em que a expressão

while for verificada, o corpo do loop será ignorado e o programa começará a ser executado na primeira instrução após o loop while. Vamos dar um exemplo para entender isso. No programa abaixo, o usuário digita o valor da variável do contador de loops, se a variável do contador for menor que 5, o loop executará e aumentará a variável do contador em um até o valor do contador ser igual a 5. Se a variável do contador for maior ou igual a 5, o loop não será executado.

```
import java.util.Scanner;

public class WhileLoopCondicionalNegativo{

    public static void main(String[] args)

    {

        int contador;

        Scanner inputDevice=new Scanner(System.in);

        System.out.print("Digite o valor do contador do loop>> ");

        contador=inputDevice.nextInt();


        System.out.println("Antes do Loop");

        while(contador<5)

        {

            System.out.println("Dentro do Loop - Contador= "+contador); contador++;

        }

        System.out.println("Depois do loop While");
```

}



}

Resultado:

```
<terminated> WhileLoopNegativeCondition [Java Application] C:\
Digite o valor do contador do loop >> 10
Antes do Loop
Depois do Loop While
```

```
<terminated> WhileLoopNegativeCondition [Java Application] C
Digite o valor do contador >> 3
Antes do Loop
Dentro do Loop - Contador = 3
Dentro do Loop - Contador = 4
Depois do Loop While
```

### Pontos importantes ao usar loops

· A condição / expressão do loop sempre pode ser verdadeira, o que torna nosso loop infinito. Essa é uma prática ruim de programação, pois pode resultar em exceção de memória. A declaração abaixo é válida, mas não é boa para ter em nosso programa.

```
while (true) {
```

while (2 < 4)

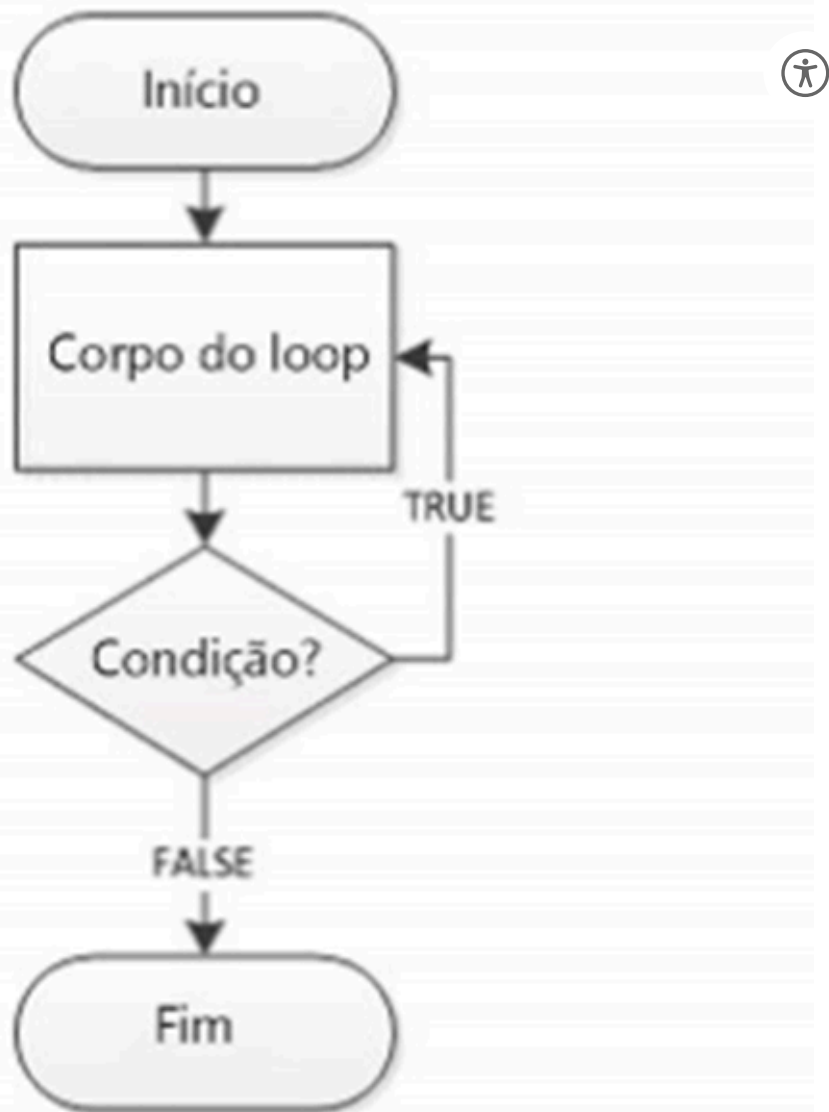


· É muito comum alterar o valor de uma variável de controle de loop adicionando 1 a ela ou incrementando a variável. No entanto, nem todos os loops são controlados adicionando 1 às vezes, podemos controlar subtraindo 1 de uma variável de controle de loop ou diminuindo-a.

### **Loop do ... while**

O loop do é semelhante ao loop while, exceto que a expressão não é avaliada até depois que o código do loop do é executado. Portanto, é garantido que o código em um loop do seja executado pelo menos uma vez. A seguir, é mostrada uma sintaxe do loop:

```
do { // Corpo do Loop Body } while (Condição);
```



Nesse caso, você deseja escrever um loop que verifique na “parte inferior” do loop após a primeira iteração. O loop do ... while verifica o valor da variável de controle do loop na parte inferior do loop após uma repetição. O código de exemplo abaixo explica o loop do... while.

```
public class DoWhileLoopDemo{
```

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

```
int i =10;
```

```
do{
```



```
i=i+10;
```



```
System.out.println("Contador do Loop =" + i);
```

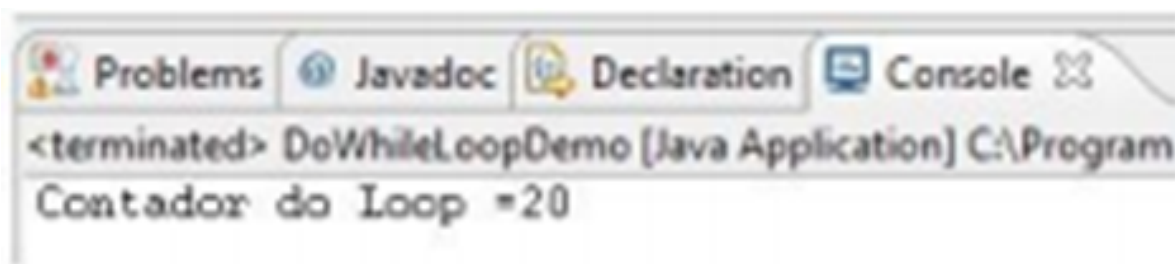
```
}
```

```
while(i<10);
```

```
}
```

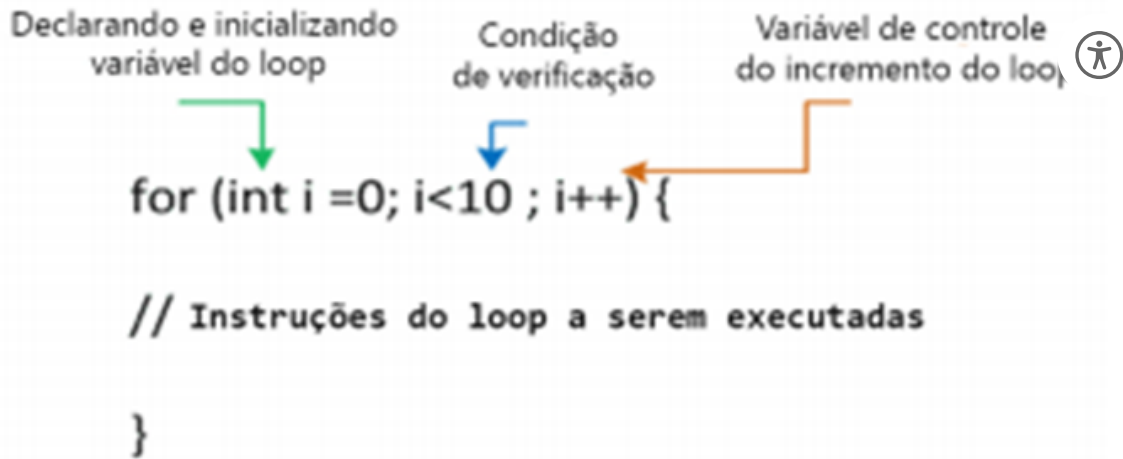
```
}
```

### Resultado:



### Loop for em Java

Um Loop For é um loop especial usado quando é necessário um número definido de iterações do loop. Embora um loop while também possa ser usado para atender a esse requisito, o Loop For fornece uma notação abreviada para esse tipo de loop. Quando você usa um Loop For, pode indicar o valor inicial da variável de controle do loop, a condição de teste que controla a entrada do loop e a expressão que altera a variável de controle do loop - tudo em um local conveniente. Abaixo está a sintaxe de um Loop For convencional.



O loop For começará com a palavra-chave For seguida por um conjunto de parênteses. Dentro dos parênteses, há três seções separadas por exatamente dois pontos e vírgulas. As três seções são geralmente usadas para o seguinte:

- Inicializando a variável de controle do loop
- Testando a variável de controle de loop
- Atualizando a variável de controle de loop

Entre os parênteses da instrução For mostrada no programa abaixo, a primeira seção anterior ao primeiro ponto-e-vírgula declara uma variável chamada count e a inicializa como 1. O programa executa essa instrução uma vez, não importa quantas vezes o corpo do loop for execute. Após a inicialização, o controle do programa passa para o meio, ou seção de teste, da instrução for. Se a expressão booleana encontrada lá for avaliada como verdadeira, o corpo do loop for será inserido. No programa, o contador é definido como 1; portanto, quando o contador <11 é testado, ele é avaliado como verdadeiro. O corpo do loop imprime o valor do contador. Se você deseja que várias instruções sejam executadas dentro do loop, elas devem ser bloqueadas dentro de um par de chaves. Depois que o corpo do loop é executado, o terço final do loop for é executado e o contador é aumentado para 2. Após a terceira seção na instrução for, o controle do programa retorna à segunda seção, onde o contador é comparado com 11 pela segunda vez. Como o contador ainda é menor que 11, o corpo executa: o contador (agora 2) é impresso e

a terceira parte alteradora do loop for é executada novamente. O contador variável aumenta para 3 e o loop for continua.



Eventualmente, quando o contador não for menor que 11 (após a impressão de 1 a 10), o loop for termina e o programa continua com quaisquer instruções que seguem o Loop For.

```
for (int contador = 1; contador <11; contador ) {System.out.println (contador);}
```

O loop For é muito útil na programação Java e amplamente utilizado em programas Java. Vamos ver mais alguns exemplos para declaração de loop.

- Inicialização de mais de uma variável colocando vírgulas entre as instruções separadas, como a seguir:

```
for (g = 0, h = 1; g <6; ++ g)
```

- Verificação de mais de uma condição usando operadores AND ou OR, como a seguir: for (g = 0; g <3 && h > 1; ++ g, h--)

- Decrementando a variável de controle do loop e verificação de alguma outra condição, como a seguir:

```
for (g = 5; g >= 1; --g)
```

- Alterando mais de um valor, como no seguinte:

```
for(g = 0; g <10; ++ g, ++ h, soma += g)
```

- Você pode deixar uma ou mais partes de um loop for vazias, embora os dois pontos e vírgulas ainda sejam necessários como espaços reservados. Por exemplo,

se x foi inicializado em uma instrução de programa anterior, você pode escrever a seguinte:



```
for (; x < 10; ++ x)
```

Vamos ver o exemplo abaixo, que imprime todos os valores divisíveis por 7 no intervalo de 1 a 100 na ordem inversa.

```
public class ForLoopDivisivelPor7{
```

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

```
for(int i=100; i>0;--i)
```

```
{
```

```
if(i%7==0)
```

```
{
```

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

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


```
}
```

```
}
```

```
}
```

```
}
```

**Loop enhanced for**

O loop enhanced for permite percorrer um array / coleção sem especificar os pontos inicial e final da variável de controle do loop. A forma geral da ver.  aprimorada para loop é mostrada aqui:

```
for (type itr-var: collection) bloco de instruções
```

Aqui, type especifica o tipo e itr-var especifica o nome de uma variável de iteração que receberá os elementos de uma coleção, uma de cada vez, do começo ao fim. Como a variável de iteração recebe valores da coleção, o tipo deve ser o mesmo que (ou compatível com) os elementos armazenados na coleção. Portanto, ao iterar sobre matrizes, o tipo deve ser compatível com o tipo base da matriz.

O exemplo abaixo mostra o uso do loop enhanced for.

```
public class EnhancedForLoopDemo{
```

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

```
int[] meuArray=new int[10];
```

```
int i=0;
```

```
// Loop for tradicional a ser preenchido
```

```
for(int k=100; k>0; k=k-10, i)
```

```
{
```

```
meuArray[i]=k;
```

```
}
```

```
// Aprimorado para os melhores elementos de loop da matriz for(int loopVal:  
meuArray)
```



```
{  
  
System.out.println(loopVal);  
  
}  
  
}  
  
}
```

## Instruções de ramificação Java

Java fornece três instruções de ramificação: break, continue e return. O Break e continue em Java são duas palavras-chave essenciais que os iniciantes precisam conhecer enquanto usam loops (loop for, loop while loop do while). A instrução break em Java é usada para interromper o loop e transfere o controle para a linha imediatamente fora do loop, enquanto continue é usado para escapar da execução atual (iteração) e transfere o controle de volta ao início do loop. Tanto break quanto a continue permitem ao programador criar construções sofisticadas de algoritmo e loop.

Veremos o exemplo da instrução break e continue em Java e alguns pontos importantes relacionados à quebra do loop usando o rótulo e a instrução break. A palavra-chave break também pode ser usada na instrução switch para interromper a escolha atual e, se não for usada, pode causar falha no switch. Tanto a instrução de break quanto a instrução de continue podem ser usadas sem rótulo ou rotuladas, embora seja muito mais comum usá-las sem rótulo.

Vamos entender a instrução break e continue sem rótulo. O programa abaixo fará a adição de todos os números pares de array até encontrar 0 ou número negativo de um array.



```
public class BreakContinueDemo{

    public static void main(String[]args){

        int[]numbers={10,23,19,34,54,23,76,39,65,24,8,0,12,55};

        int sum =0;

        for(int i=0; i<numbers.length; i++){

            if(numbers[i]<=0){

                System.out.println("Break porque número =

                "+numbers[i]);

                break;

            }elseif(numbers[i]%2!=0){

                System.out.println("Número ímpar encontrado

                na matriz, ignorando o número "+numbers[i]);

                continue;

            }else

                sum= sum +numbers[i];

        }

        System.out.println("Soma de todos os números = "+ sum);
```

}



}

## Instruções rotuladas

Embora muitas instruções possam ser rotuladas, é mais comum usar rótulos com instruções de loop como `for` ou `while`, em conjunto com instruções de `break` e `continue`. Uma instrução rotulada consiste em um identificador válido que termina com dois pontos (:).

Você precisa entender a diferença entre `break` e `continue` rotulado e não rotulado. Os rotulados são necessários apenas nas situações em que você possui um loop aninhado e precisam indicar de quais loops aninhados você deseja interromper ou de quais loops aninhados deseja continuar na próxima iteração. Uma instrução `break` sairá do loop rotulado, em oposição ao loop mais interno, se a palavra chave `break` for combinada com um rótulo. Aqui está um programa de exemplo que usa `continue` para imprimir uma tabela de multiplicação triangular de 0 a 9.



```
public class LabeledBreakContinueDemo{ Ⓢ

    public static void main(String[]args){

        int breaklimit=9;

        outer:for(int i =0;; i++){

            for(int j =0; j <10; j++){

                if(j > i){

                    System.out.println();

                    continue outer;

                }

                System.out.print(" "+(i * j));

            }

            if(i==breaklimit){

                break outer;

            }

        }

    }

}
```

```
System.out.println();
```



```
}
```

```
}
```

## A instrução return

A última instrução de controle é return. Return é usada para retornar explicitamente de um método. Ou seja, faz com que o controle do programa seja transferido de volta ao chamador do método. A qualquer momento em um método, a instrução return pode ser usada para fazer com que a execução volte ao chamador do método. Assim, return finaliza imediatamente o método em que é executada. O exemplo a seguir ilustra esse ponto. Abaixo, o programa main () está chamando o método e checaPar () é chamado o método. A execução do método checaPar () termina quando a instrução return é encontrada.

```
public class ReturnDemo{
```

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

```
for(int k =25; k<31; k++){
```

```
new ReturnDemo().checaPar(k);
```

```
}
```

```
}
```

```
public boolean checaPar(int a){
```

```
if(a%2==0){
```

```
System.out.println(a + " é par");
```



```
return true;
```

```
}
```

```
System.out.println(a + " é ímpar");
```

```
}
```

```
}
```

Resultado:

```
return false
```

## Atividade Extra

Vídeo: Estrutura de repetição em Java

Link para o vídeo: <https://www.youtube.com/watch?v=2fawKjR8d4c>



## Referências Bibliográficas

BARNES, D. J.; KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej**. 4.ed. Pearson: 2009.

FELIX, R. (Org.). **Programação orientada a objetos**. Pearson: 2017.

MEDEIROS, L. F. de. **Banco de dados: princípios e prática**. Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java**. Disponível em: < <https://docs.oracle.com/en/java/> >.

**Ir para exercício**