

Slide 1



CONTROLANDO O FLUXO

FUNDAMENTOS DE PROGRAMAÇÃO

✓ Condicionais

As linguagens de programação têm instruções que permitem que você tome decisões sobre o que fazer a seguir.

Algumas dessas declarações permitem que você repita uma determinada atividade várias vezes.

Este capítulo discute várias dessas declarações, bem como explora algumas questões relacionadas à comparação de dados e objetos.

CONTROLANDO O FLUXO

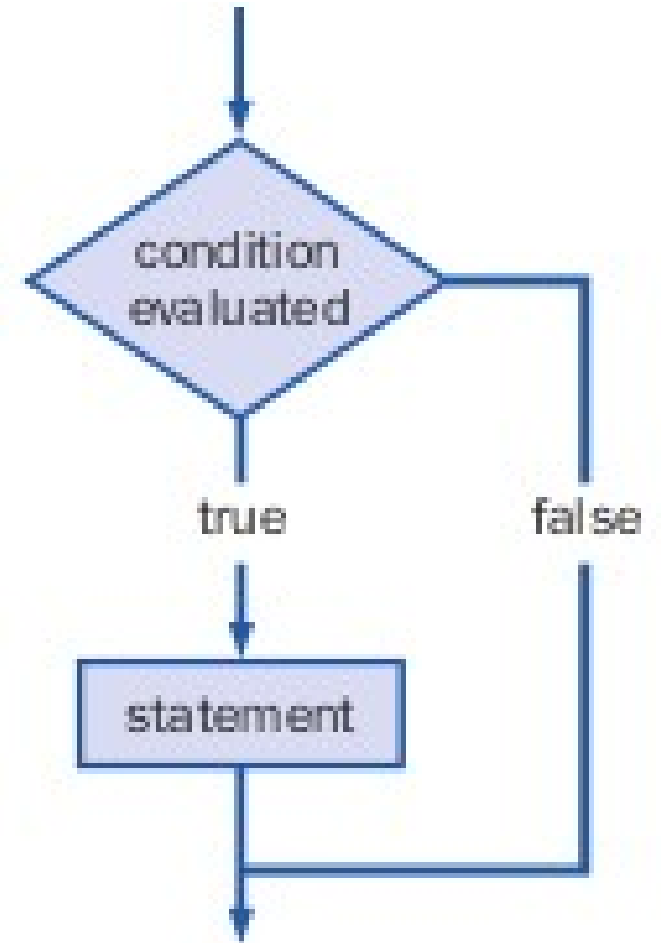
✓ A Declaração If

Uma instrução if consiste na palavra reservada `se` seguida por uma expressão booleana, seguida por uma instrução.

A condição está entre parênteses e deve ser avaliada como verdadeira ou falsa.

Se a condição for verdadeira, a instrução será executada e, em seguida, o processamento continuará com qualquer instrução a seguir.

Se a condição for falsa, a instrução controlada pela condição será ignorada e o processamento continuará imediatamente com qualquer instrução a seguir.



CONTROLANDO O FLUXO

Considere o seguinte exemplo de uma instrução if:

```
IF ( total > montante)  
    total = total + (montante + 1);
```

Neste exemplo, se o valor no total for maior do que o valor no valor, a instrução de atribuição será executada; caso contrário, a instrução de atribuição será ignorada.

Observe que a instrução de atribuição neste exemplo é recuada sob a linha de cabeçalho da instrução if. Isso comunica a um leitor humano que a declaração de atribuição faz parte da declaração if; implica que a instrução if rege se a instrução de atribuição será executada.

✓ A Declaração If

```
import java.util.Scanner;
public class Age{
    public static void main(String[] args){
        final int MINOR = 21;
        Scanner scan = new Scanner(System.in);
        System.out.print("Digite sua idade: ");
        int age = scan.nextInt();
        System.out.println("Você digitou: " + age);
        if (age < MINOR)
            System.out.println("Juventude é uma coisa maravilhosa. Aproveitar.");
        System.out.println("A idade é um estado de espírito.");
    }
}
```

✓ A declaração if-else

Às vezes, queremos fazer uma coisa se uma condição é verdadeira e outra coisa se essa condição é falsa. Para lidar com esse tipo de situação, podemos adicionar uma cláusula else a uma instrução if, tornando-a uma instrução if-else. Segue-se um exemplo de uma instrução if-else:

```
if (altura <= MAX)
    ajuste = 0;
else
    ajuste = MAX - height;
```

Uma instrução if-else permite que um programa faça uma coisa se uma condição for verdadeira e outra coisa se a condição for falsa.

✓ A declaração if-else

Uma instrução if testa a Expressão booleana e, se a Expressão for verdadeira, executa a primeira Instrução. A cláusula else opcional identifica a Instrução que deve ser executada se a Expressão for falsa.

Exemplos:

```
if (total < 7)
    System.out.println("Total inferior a 7.");
if (firstCh != 'a')
    contagem++;
else
    contagem = contagem / 2;
```

CONTROLANDO O FLUXO

If Statement

The diagram illustrates an if-else statement with the following code and annotations:

```
if (total <= cash)
    cash = cash - total;
else
{
    system.out.println("Insufficient cash.");
    total = 0;
}
```

Annotations:

- boolean condition**: Points to the condition `total <= cash`.
- executed if the condition is true**: Points to the block of code `cash = cash - total;`.
- block statement**: Points to the `else` block.
- executed if the condition is false**: Points to the code `system.out.println("Insufficient cash.");` and `total = 0;` inside the `else` block.

✓ Instruções se aninhado

A instrução executada como resultado de uma instrução if pode ser outra instrução if.

Essa situação é chamada de se aninhado. Permite-nos tomar outra decisão depois de determinar os resultados de uma decisão anterior.

O programa chamado MinOfThree, usa instruções if aninhadas para determinar o menor de três valores inteiros inseridos pelo usuário.

Trace cuidadosamente a lógica do programa MinOfThree, usando vários conjuntos de entrada com o valor mínimo em todas as três posições, para ver como ele determina o menor valor.

Uma situação importante surge com as instruções if aninhadas. Pode parecer que uma cláusula else após um aninhado if poderia se aplicar a qualquer uma das instruções if.

✓ Considere o seguinte exemplo:

A instrução executada como resultado de uma instrução if pode ser outra instrução if.

Essa situação é chamada de se aninhado. Permite-nos tomar outra decisão depois de determinar os resultados de uma decisão anterior.

O programa chamado MinOfThree, usa instruções if aninhadas para determinar o menor de três valores inteiros inseridos pelo usuário.

Trace cuidadosamente a lógica do programa MinOfThree, usando vários conjuntos de entrada com o valor mínimo em todas as três posições, para ver como ele determina o menor valor.

Uma situação importante surge com as instruções if aninhadas. Pode parecer que uma cláusula else após um aninhado if poderia se aplicar a qualquer uma das instruções if.

CONTROLANDO O FLUXO

```
public class MinOfThree{
    public static void main(String[] args){
        int num1, num2, num3, min = 0;
        Scanner scan = new Scanner(System.in);
        System.out.println("Insira três inteiros: ");
        num1 = scan.nextInt();
        num2 = scan.nextInt();
        num3 = scan.nextInt();
        if (num1 < num2)
            if (num1 < num3)
                min = num1;
            else
                min = num3;
        else
            if (num2 < num3)
                min = num2;
            else
                min = num3;
        System.out.println("Valor mínimo: " + min);
    }
}
```

✓ A declaração switch

Outra instrução condicional em Java, chamada instrução switch, faz com que o programa em execução siga um dos vários caminhos com base em um único valor.

Também discutimos a instrução break nesta seção porque ela geralmente é usada com uma instrução switch.

A instrução switch avalia uma expressão para determinar um valor e, em seguida, faz a correspondência desse valor com um dos vários casos possíveis. Cada caso tem declarações associadas a ele. Depois de avaliar a expressão, o controle salta para a instrução associada ao primeiro caso que corresponde ao valor.

Considere o seguinte exemplo:

✓ A declaração switch

```
switch (idChar){  
  case 'A':  
    aCount = aCount + 1;  
    break;  
  case 'B':  
    bCount = bCount + 1;  
    Break;  
  case 'C':  
    cCount = cCount + 1;  
    break;  
  default:  
    System.out.println("Erro no caráter de identificação.");  
}
```

CONTROLANDO O FLUXO

Primeiro, avalia-se a expressão. Neste exemplo, a expressão é uma variável char simples. Em seguida, a execução é transferida para a primeira instrução identificada pelo valor do caso que corresponde ao resultado da expressão.

Portanto, se idChar contiver um 'A', a variável aCount será incrementada. Se contiver um 'B', o caso para 'A' é ignorado e o processamento continua onde bCount é incrementado.

Se nenhum valor de maiúscula e minúscula corresponder ao da expressão, a execução continuará com o caso padrão opcional, indicado pela palavra reservada default.

Se nenhum caso padrão existir, nenhuma instrução na instrução switch será executada e o processamento continuará com a instrução após a instrução switch.

Muitas vezes, é uma boa ideia incluir um caso padrão, mesmo que você não espere que ele seja executado.

CONTROLANDO O FLUXO

Quando uma instrução `break` é encontrada, o processamento salta para a instrução após a instrução `switch`.

Uma instrução `break` é geralmente usada para quebrar cada caso de uma instrução `switch`.

Sem uma instrução de interrupção, o processamento continua no próximo caso do `switch`.

Portanto, se a instrução `break` no final do caso 'A' no exemplo anterior não estivesse lá, tanto a variável `aCount` quanto a variável `bCount` seriam incrementadas quando `idChar` contivesse um 'A'.

Normalmente, queremos executar apenas um caso, por isso uma instrução de quebra é quase sempre usada. Ocasionalmente, porém, o recurso "passar através" é útil.

✓ A declaração switch

A instrução switch avalia a expressão inicial e corresponde seu valor com um dos casos.

O processamento continua com a Declaração correspondente a esse caso. O caso padrão opcional será executado se nenhum outro caso corresponder.

CONTROLANDO O FLUXO

```
switch (numValues){  
    case 0:  
        System.out.println("No values were entered.");  
        break;  
    case 1:  
        System.out.println("One value was entered.");  
        break;  
    case 2:  
        System.out.println("Two values were entered.");  
        break;  
    default:  
        System.out.println("Too many values were entered.");  
}
```

Observe que a condição booleana implícita de uma instrução switch é baseada na igualdade. A expressão no início da instrução é comparada com cada valor de caso para determinar qual é igual. Uma instrução switch não pode ser usada para determinar outras operações relacionais (como menos que), a menos que algum processamento preliminar seja feito. Por exemplo, o programa GradeReport que imprime um comentário com base em uma nota numérica inserida pelo usuário.

CONTROLANDO O FLUXO

```
public class GradeReport{  
    public static void main(String[] args) {  
        int nota, categoria;  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Insira uma nota numérica (0 a 100): ");  
        grade = scan.nextInt();  
        categoria = nota / 10;  
        System.out.print("Essa nota é ");  
        switch (categoria) {  
            case 10:  
                System.out.println("uma pontuação perfeita. Parabéns.");  
                break;
```

CONTROLANDO O FLUXO

```
case 9:
    System.out.println("bem acima da média. Excelente.");
    break;
case 8:
    System.out.println("acima da média. Bom trabalho.");
    break;
case 7:
    System.out.println("média.");
    break;
case 6:
    System.out.print(" abaixo da média. Por favor, veja o");
    System.out.println("instrutor para assistência.");
    Break;
default:
    System.out.println("não passa.");}}
}
```

Observe que qualquer instrução switch pode ser implementada como um conjunto de instruções if aninhadas.

No entanto, as instruções aninhadas se rapidamente se tornam difíceis de entender para um leitor humano e são propensas a erros para implementar e depurar.

Mas como um switch pode avaliar apenas a igualdade, aninhamos se as declarações às vezes forem necessárias. Depende da situação.

✓ A Declaração "while"

Uma instrução while executa a mesma instrução repetidamente até que sua condição se torne falsa.

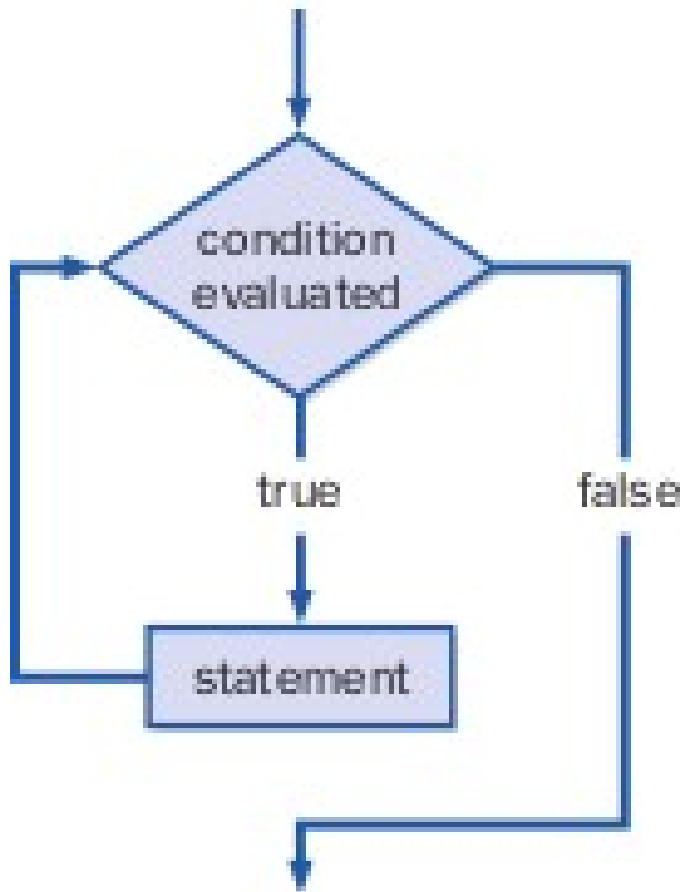
uma instrução de repetição (ou loop) nos permite executar outra instrução várias vezes.

Uma instrução while é um loop que avalia uma condição booleana assim como uma instrução if e executa uma instrução (chamada corpo do loop) se a condição for verdadeira.

No entanto, ao contrário da instrução if, depois que o corpo é executado, a condição é avaliada novamente. Se ainda for verdade, o corpo é executado novamente.

Esta repetição continua até que a condição se torne falsa; em seguida, o processamento continua com a instrução após o corpo do loop while.

CONTROLANDO O FLUXO



O loop a seguir imprime os valores de 1 a 5. Cada iteração através do loop imprime um valor e, em seguida, incrementa o contador.

```
int count = 1;
while (count <= 5) {
    System.out.println(count);
    count++;
}
```

CONTROLANDO O FLUXO

Observe que o corpo do loop while é um bloco contendo duas instruções. O bloco inteiro é repetido em cada iteração do loop.

O loop while executa repetidamente a instrução especificada, desde que a expressão booleana seja verdadeira.

A Expressão é avaliada primeiro; portanto, a instrução pode não ser executada.

A Expressão é avaliada novamente após cada execução da Instrução até que a Expressão se torne falsa.

CONTROLANDO O FLUXO

```
while (total > max)
{
total = total / 2;
System.out.println("Total actual: " + total);
}
```

While Loop

boolean condition

```
while (input <= 0)
{
    System.out.println("Input must be positive.")
    input = scan.nextInt();
}
```

executed repeatedly as long
as the condition is true

✓ A Declaração do

A instrução `do` é semelhante à instrução `while` na medida em que executa o corpo do loop até que uma condição se torne falsa.

No entanto, ao contrário do loop `while`, cuja condição é avaliada antes do corpo ser executado, a condição de um loop `do` é avaliada depois que o corpo do loop é executado. Sintaticamente, a condição em um loop `do` é escrita após o corpo do loop para refletir esse processamento.

O corpo de um loop `do` é sempre executado pelo menos uma vez, enquanto que com um loop `while`, o corpo pode não ser executado (se a condição for inicialmente falsa).

✓ A Declaração do

```
int count = 0;  
do  
{  
    count++;  
    System.out.println(count);  
}  
while (count < 5);
```

O loop do executa repetidamente a instrução especificada, desde que a expressão booleana seja verdadeira.

A instrução é executada pelo menos uma vez e, em seguida, a expressão é avaliada para determinar se a instrução deve ser executada novamente.

✓ Exemplo:

```
do
{
System.out.print("Digite uma palavra:");
palavra = scan.next();
System.out.println(palavra);
}
while (!palavra.equals("sair"));
```

Um loop do começa simplesmente com a palavra reservada do.

O corpo do loop do continua até a cláusula while que contém a condição booleana que determina se o corpo do loop será executado novamente.

✓ A Declaração para ou for

A instrução for é geralmente usada quando um loop será executado um determinado número de vezes.

A instrução while e a instrução do são boas para usar quando você não sabe inicialmente quantas vezes deseja executar o corpo do loop. A instrução for é outra instrução de repetição que é particularmente adequada para executar o corpo de um loop um número específico de vezes que pode ser determinado antes que o loop seja executado.

O código a seguir imprime os números de 1 a 5 usando um loop for, assim como fizemos usando um loop while e um loop do em exemplos anteriores.

```
for (int count=1; count <= 5; count++)  
    System.out.println(count);
```

✓ A Declaração para ou for

O cabeçalho de um loop for contém três partes separadas por ponto-e-vírgula. Antes do loop começar, a primeira parte do cabeçalho, chamada de inicialização, é executada.

A segunda parte do cabeçalho é a condição booleana, que é avaliada antes do corpo do loop (como o loop while). Se true, o corpo do loop é executado, seguido pela execução da terceira parte do cabeçalho, que é chamado de incremento.

Observe que a parte de inicialização é executada apenas uma vez, mas a parte de incremento é executada após cada iteração do loop.

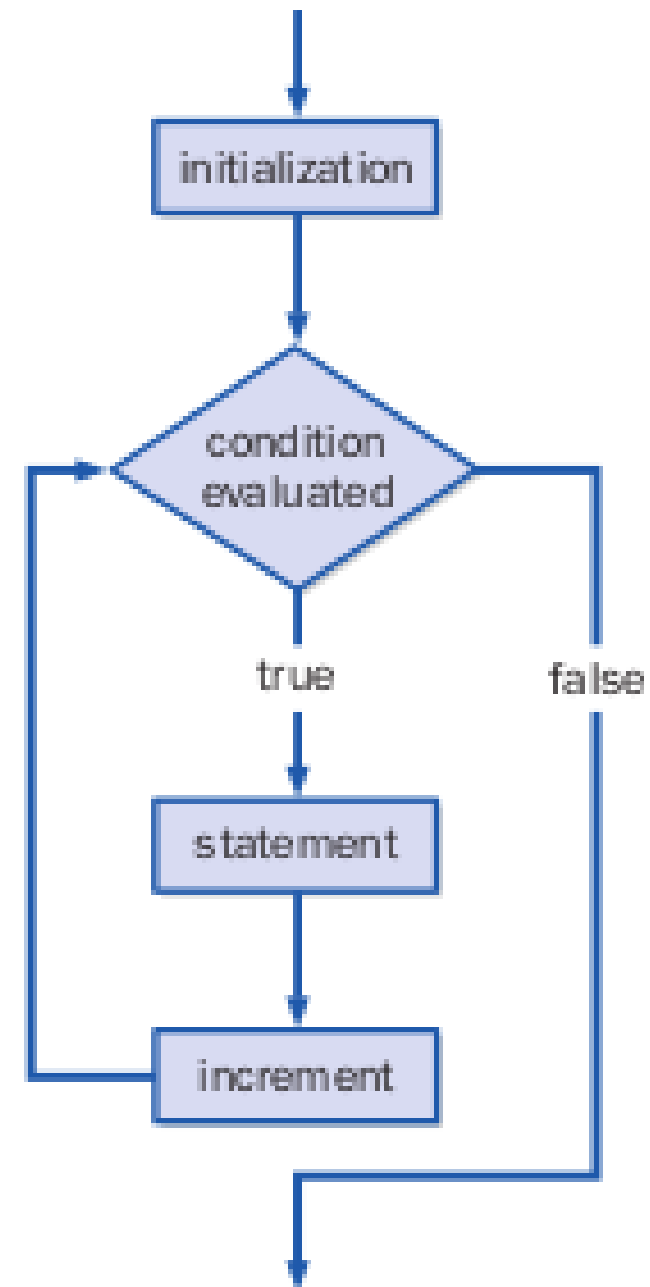
CONTROLANDO O FLUXO

✓ A Declaração para ou for

A instrução for executa repetidamente a instrução especificada, desde que a expressão booleana seja verdadeira.

A parte For Init do cabeçalho é executada apenas uma vez, antes do início do loop.

A parte For Update é executada após cada execução da instrução.



CONTROLANDO O FLUXO

✓ Exemplos:

```
for (int value=1; value < 25; value++)  
    System.out.println(value + " Quadrado é " + value*value);
```

```
for (int num=40; num > 0; num-=3)  
    sum = sum + num;
```

✔ Declaração de quebra

A instrução `break` só pode ser usada dentro das instruções `switch`, `for`, `while` e `do-while`.

Você já viu como ele pode ser usado dentro da instrução `switch`, então vamos mostrar como usá-lo em todos os outros.

A quebra de um loop `for`, `while` ou `do-while` pode ser feita usando a instrução `break`, mas deve ser controlada por uma condição de saída, caso contrário, nenhuma etapa será executada.

✓ Declaração de quebra

```
public class BreakingForDemo {  
    public static final int arr[] = {5, 1, 4, 2, 3};  
    public static void main(String... args) {  
        for (int i = 0; i < arr.length ; ++i) {  
            if (i == 3) {  
                System.out.println("Bye bye!");  
                break;  
            }  
            System.out.println("arr[" + i + "] = " + arr[i]);  
        }  
    }  
}
```

Se tivermos um caso de loops aninhados, um rótulo pode ser usado para decidir a instrução de looping a ser quebrada.

✔ Declaração de quebra

```
public class BreakingNestedForLoopDemo {  
    public static final int arr[] = {5, 1, 4, 2, 3};  
    public static void main(String... args) {  
        for (int i = 0; i < 2; ++i) {  
            HERE: for (int j = 0; j < 2; ++j) {  
                for (int k = 0; k < 2; ++k) {  
                    if (i == j && j == k) {  
                        break HERE;  
                    }  
                    _x0007_System.out.println("(i, j, k) = (" + i + ", " + j + ", " + k + ")");  
                }  
            }  
        }  
    }  
}
```

✓ A Declaração continuar

A instrução continue não quebra um loop, mas pode ser usada para ignorar certas etapas com base em uma condição.

Essencialmente, a instrução continue interrompe a execução da etapa atual do loop e passa para a próxima, então você pode dizer que essa instrução continua o loop.

Vamos continuar experimentando o exemplo de travessia de matriz e, desta vez, vamos pular a impressão dos elementos com índices ímpares usando a instrução continue.

✓ A Declaração continuar

Ignorando elementos de impressão com índices ímpares usando uma instrução for Loop e continuar

```
public class ContinueForDemo {  
    public static final int arr[] = {5, 1, 4, 2, 3};  
    public static void main(String... args) {  
        for (int i = 0; i < arr.length; ++i) {  
            if (i % 2 != 0) {  
                continue;  
            }  
            System.out.println("arr[" + i + "] = " + arr[i]);  
        }  
    }  
}
```

THANK YOU!

CONTACT US AT:

 **Nelson Norte**

 **nnorte@ucm.ac.mz**

 **+258 82 88 48 766**

