Programação em Java

OXETECH LAB

Módulo 2

Estrutura de Controle

- Introdução às Estrutura de Controle
- Estrutura de Decisão
  - Condicional if
  - Condicional if-else
  - Condicional if-else-if
  - Condicional switch
- Estruturas de Repetição
  - Laço for
  - Laço while
  - Laço do-while
- Palvras-chave de Controle
  - break
  - continue

Introdução às Estrutura de Controle

As estruturas de controle de fluxo são blocos fundamentais em qualquer linguagem de programação. Elas permitem controlar a ordem em que as instruções são executadas, determinando se um conjunto de instruções será executado ou repetido, dependendo de condições específicas. Essas estruturas são essenciais para a criação de programas dinâmicos e inteligentes, já que possibilitam a execução de diferentes caminhos e ações baseadas nas decisões do usuário ou do sistema.

Em Java, as estruturas de controle podem ser divididas em duas grandes categorias: **estruturas de decisão** e **estruturas de repetição.**

Por que Estruturas de Controle São Importantes ?

Sem estruturas de controle, os programas seriam executados de maneira linear, ou seja, uma instrução após a outra, sem nenhuma tomada de decisão ou repetição de ações. Isso limitaria severamente a utilidade dos programas. Com o uso de controle de fluxo, é possível:

- **Criar programas dinâmicos** que respondem de maneira diferente a diferentes entradas do usuário.
- **Executar tarefas repetitivas de forma eficiente**, economizando tempo e recursos.
- **Dividir problemas complexos em partes menores**, facilitando a solução e manutenção do código.

Estruturas de Decisão

As **estruturas de decisão** são usadas quando é necessário fazer escolhas entre diferentes caminhos de execução dentro do programa. Elas permitem que o programa avalie uma condição (geralmente uma expressão booleana) e, com base no resultado, decida qual bloco de código será executado.

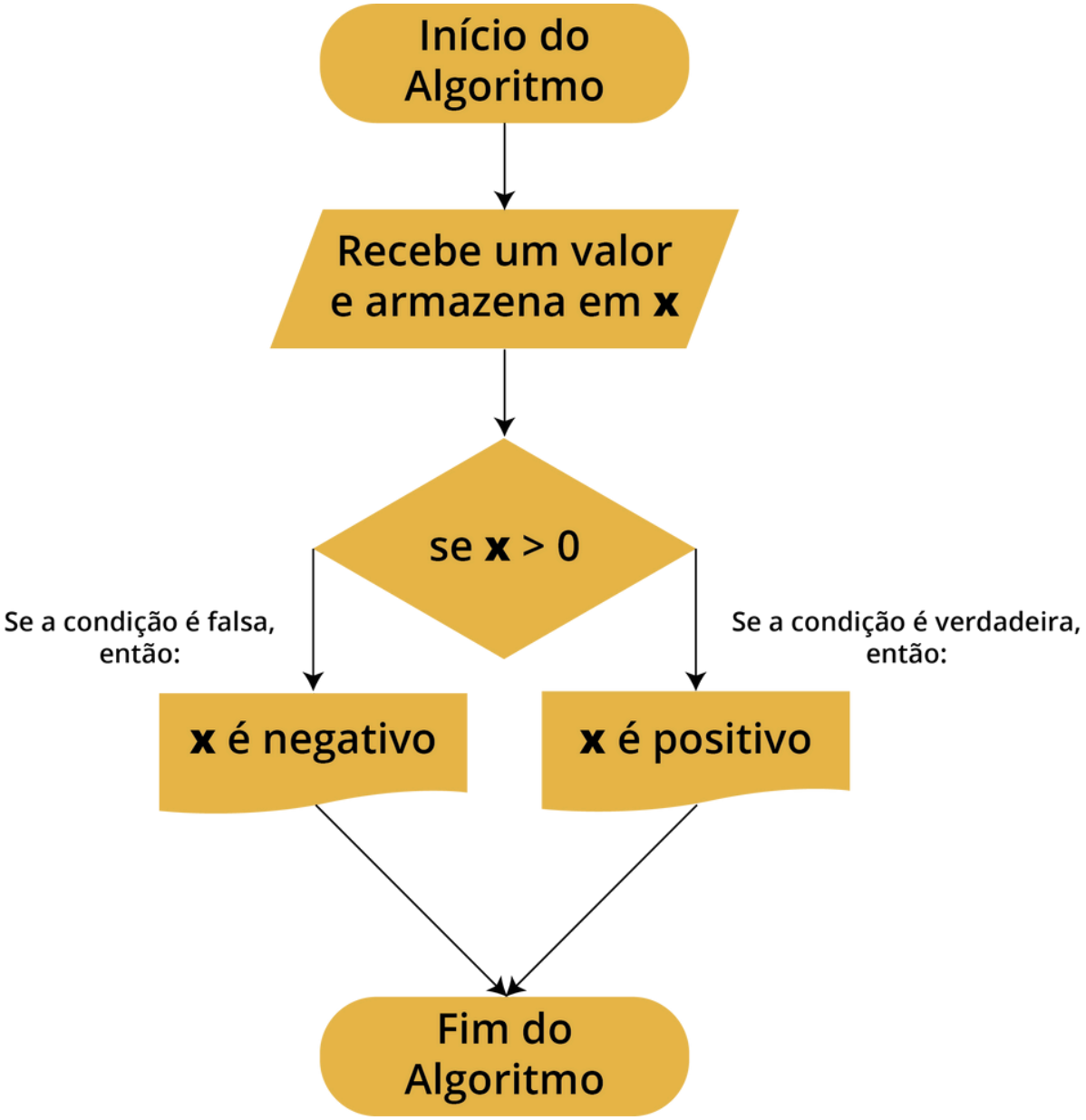
As principais estruturas de decisão em Java são:

- **If:** Avalia uma condição booleana. Se a condição for verdadeira, um bloco de código é executado.
- **If-Else:** Similar ao if, mas também permite uma alternativa. Se a condição for falsa, o bloco de código associado ao else é executado.
- **If-Else-If:** Permite a verificação de múltiplas condições de forma sequencial, executando o bloco de código correspondente à primeira condição verdadeira.
- **Switch:** Oferece uma alternativa mais clara para decisões que envolvem múltiplos valores possíveis para uma variável. Em vez de encadear vários if-else, o switch avalia a variável e executa o bloco de código associado ao valor correspondente.

Essas estruturas de decisão são amplamente usadas para resolver problemas em que o programa precisa tomar decisões com base em condições, como por exemplo, exibir diferentes mensagens ao

usuário, calcular valores ou até mesmo determinar o próximo passo de execução em um jogo ou sistema.

A figura abaixo ilustra uma estrutura de decisão onde, dependendo do valor digitado pelo usuário, o código seguirá um caminho se o valor for positivo e outro caminho se for negativo:



Condicional if

A estrutura condicional **if**, em português se, permite que o programa execute um bloco de código apenas se uma condição for verdadeira. Se a condição for falsa, o bloco é ignorado e o programa continua com o restante das instruções.

Sintaxe:

```
if (condição) {
   // Bloco a ser executado se a condição for true
 }
```

Exemplo:

```
int idade = 20;
if (idade >= 18) {
   System.out.println("Você é maior de idade.");
}
```

No exemplo acima, a mensagem "Você é maior de idade" será impressa apenas se a condição idade >= 18 for verdadeira. Se a idade fosse menor que 18, nada aconteceria.

Observações: A condição dentro do if pode ser qualquer expressão que resulte em um valor booleano (verdadeiro ou falso), podem ser usadas expressões mais complexas e até funções dentro da condição e o bloco de código dentro do if pode conter múltiplas linhas de código.

Também é possível usar o condicional if de forma aninhada, ou seja, um dentro do outro. Essa abordagem permite verificar múltiplas condições de forma hierárquica, onde cada condição é avaliada somente se a condição anterior for verdadeira.

Sintaxe:

```
if (condição 1) {
   // instruções para condição 1 verdadeira
   if (condição 2) {
      // instruções para condição 2 verdadeira
   }
}
```

Exemplo:

```
int numero = 10;
if (numero > 0) {
   System.out.println(“O número é positivo.”);
   if (numero % 2 == 0) {
      System.out.println(“O número é par.”);
   }
}
```

Neste exemplo, o programa primeiro verifica se numero é maior que 0. Se for, ele imprime "O número é positivo." e então verifica se numero é par (numero % 2 == 0). Se ambas as condições forem verdadeiras, ele também imprime "O número também é par."

Observações: Comandos if aninhados podem aumentar a complexidade do código, por isso é importante mantê-los claros e bem organizados. É possível aninhar múltiplos níveis de if, mas isso pode afetar a legibilidade do código.

Condicional if-else

A estrutura **if-else** é uma extensão do if e permite definir uma ação alternativa caso a condição seja falsa. Isso garante que o programa execute um bloco de código em qualquer caso, seja quando a condição é verdadeira ou falsa.

Sintaxe:

```
if (condição) {
   // Bloco a ser executado se a condição for true
} else {
   // Bloco a ser executado se a condição for false
}
```

Exemplo:

```
int idade = 16;
if (idade >= 18) {
   System.out.println("Você é maior de idade.");
} else {
   System.out.println("Você é menor de idade.");
}
```

Se a idade for maior ou igual a 18, a primeira mensagem será exibida. Caso contrário, a segunda mensagem será exibida.

Observações: O bloco de código dentro do else é executado apenas se a condição do if for falsa, o else não pode existir sem um if correspondente e o if-else é útil para garantir que uma das duas alternativas seja executada, proporcionando um controle de fluxo mais robusto.

O condicional if-else também pode ser usada de forma aninhada, permitindo verificar múltiplas condições em sequência, fazendo com que diferentes blocos de código sejam executados com base em várias condições.

Sintaxe:

```
if (condição 1) {
   // Caso condição 1 true
   if (condição 2) {
      // Caso condição 1 true e condição 2 true
    } else {
    // Caso condiçõa 1 true e condição 2 false
    }
} else {
 // Caso condição 1 false
}
```

Exemplo:

```
int numero = 10;
if (numero != 0) {
   System.out.println(“O número não é zero.”);
   if (numero > 0) {
      System.out.println(“O número é positivo.”);
   } else {
      System.out.println(“O número é negativo.”);
} else {
   System.out.println(“O número é zero.”);
}
```

Neste exemplo, o programa primeiro verifica se numero não é zero. Se for verdadeiro, ele imprime "O número não é zero." e então verifica se numero é maior que 0. Se numero for positivo, ele imprime "O número é positivo." Se numero não for positivo (ou seja, for negativo), ele imprime "O número é negativo." Se a primeira condição for falsa (ou seja, se numero for zero), ele imprime "O número é zero."

Observações: Aninhar múltiplos comandos if-else pode aumentar a complexidade do código, então é importante manter o código claro e bem organizado. Use indentação adequada para melhorar a legibilidade do código.

Condicional if-else-if

O **if-else-if** é uma forma de encadear múltiplas condições. Ele permite verificar várias expressões booleanas em sequência e executar o bloco de código correspondente à primeira condição verdadeira. Se nenhuma condição for verdadeira, o bloco do else final (opcional) será executado.

Sintaxe:

```
if (condição1) {
   // Caso a condição 1 for true
} else if (condição2) {
   // Caso a condição 2 for true
} else {
   // Caso nenhuma condição for true
}
```

Exemplo:

```
int nota = 85;
if (nota >= 90) {
   System.out.println(“Excelente”);
} else if (nota >= 70) {
   System.out.println(“Bom”);
} else {
   System.out.println(“Precisa melhorar”);
}
```

Neste exemplo, se a nota for maior ou igual a 90, a mensagem "Excelente" será impressa. Se for entre 70 e 89, a mensagem "Bom" será exibida. Caso contrário, "Precisa melhorar" será exibido.

Observações: As condições são avaliadas em sequência, e a execução para na primeira condição verdadeira. O bloco else é opcional, mas é uma boa prática incluí-lo para lidar com casos onde todas as outras condições são falsas.

Condicional switch

A estrutura **switch** é útil quando se deseja testar uma única variável contra múltiplos valores possíveis. Ela é frequentemente usada como uma alternativa ao if-else-if quando temos muitos casos baseados em igualdade de valor.

Sintaxe:

```
switch (variável) {
   case valor1:
      // Bloco de código para o caso valor1
   break;
   case valor2:
      // Bloco de código para o caso valor2
   break;
   default:
      // Bloco de código se nenhum valor corresponder
}
```

Exemplo:

```
int diaSemana = 3;
switch (diaSemana) {
   case 1:
      System.out.println(“Domingo”);
      break;
   case 2:
      System.out.println(“Segunda”);
      break;
   case 3:
      System.out.println(“Terça”);
      break;
   case 4:
      System.out.println(“Quarta”);
      break;
   default:
      System.out.println(“Dia inválido);
}
```

Aqui, se diaDaSemana for igual a 3, o programa exibirá "Terça-feira". Se o valor não corresponder a nenhum dos casos, o bloco default será executado.

Vamos entender melhor como o swtich funciona, o programa avalia a expressão dentro dos parênteses após switch. O valor da expressão é comparado sequencialmente com cada case.

Quando uma correspondência é encontrada, o bloco de código associado ao case correspondente é executado.
A instrução **break** termina a execução do bloco de código do case atual e impede que o controle passe para o próximo case.
Se nenhum dos cases corresponder ao valor da expressão, o bloco de código dentro de default é executado.

Observações: Cada **case** deve terminar com uma instrução break para evitar que o controle passe para o próximo case (efeito conhecido como "fall through"). O bloco **default** é opcional, mas é uma boa prática incluí-lo para lidar com valores inesperados. O comando **switch** é mais eficiente que uma série de comandos if-else quando se trata de comparar a mesma expressão com vários valores diferentes.

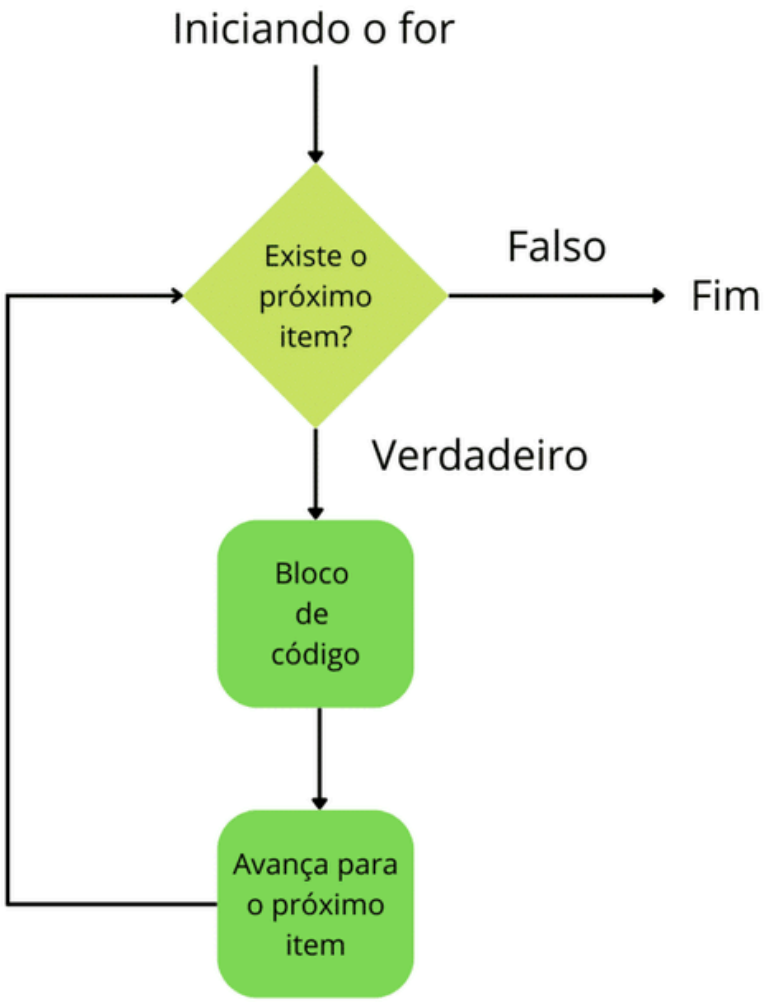
Conslusão:

As estruturas de decisão são essenciais para criar programas dinâmicos que respondem de acordo com entradas e condições. O uso correto dessas estruturas permite que o desenvolvedor escreva programas mais inteligentes e flexíveis, com a capacidade de tomar decisões baseadas em lógica. Ao dominar o if, if-else, if-else-if e switch, você estará preparado para construir algoritmos que possam lidar com diferentes cenários e fornecer resultados apropriados com base nas condições impostas.

Estruturas de Repetição

As **estruturas de repetição** são usadas em programação para executar repetidamente um bloco de código, até que uma condição seja satisfeita ou enquanto uma condição é verdadeira. Elas permitem que os programas sejam mais eficientes e reduzem a duplicação de código, pois podem executar a mesma operação várias vezes sem a necessidade de reescrever as instruções. Em java, existem três principais estruturas de repetição: **for**, **while** e **do-while**. Cada uma com características diferentes.

Adiagrama abaixo ilustra o funcionamento de um estrutura de repetição, fazendo checagem de uma lista de item até que não haja mais itens na lista.



Laço for

A estrutura **for** é geralmente usada quando você sabe de antemão quantas vezes deseja que o bloco de código seja executado. Ela é mais compacta que a estrutura while e do-while, pois agrupa em uma única linha a inicialização, a condição e a atualização de uma variável de controle.

Sintaxe:

```
for (inicialização; condição; atualização) {
   // Bloco de código a ser repetido
}
```

Como funciona ?

- I**nicialização:** Executada uma vez no início do laço, usada para definir e inicializar variáveis de controle.
- **Condição:** Avaliada antes de cada iteração. Se a condição for verdadeira, o bloco de código é executado.
- **Atualização:** Executado após cada iteração, geralmente usado para atualizar a variável de controle.

Exemplo:

```
for (int i = 0; i < 5; i ++) {
   System.out.println(“Contagem ” + i);
}
```

Define e inicializa a variável i com o valor 0. Esta parte do código é executada apenas uma vez, no início do laço.
Antes de cada iteração, a condição i < 5 é avaliada. Se a condição for verdadeira, o bloco de código dentro do laço é executado. Se a condição for falsa, o laço termina e a execução continua após o bloco de código do laço.
Após cada iteração, a expressão i++ é executada, incrementando i em 1. Depois do incremento, o controle volta para a avaliação da condição para a próxima iteração.
O laço for neste exemplo executa o bloco de código cinco vezes, imprimindo os valores de 0 a 4.

Laço while

A estrutura **while** é a mais básica entre as estruturas de repetição. Ela executa um bloco de código repetidamente enquanto uma condição for verdadeira. Se a condição for falsa desde o início, o bloco de código não será executado nem uma única vez.

Sintaxe:

```
while (condição) {
   // Bloco de código a ser repetido
}
```

Como funciona ?

A condição é avaliada antes de cada iteração. Se for verdadeira, o bloco de código é executado. Se a condição for falsa, a execução do laço termina e o controle passa para a instrução seguinte.

Exemplo:

```
int i = 0;

while (i < 5) {
   System.out.println(“Contagem ” + i);
   i++;
}
```

Declaramos e inicializamos a variável i com o valor 0. Essa variável será usada como a variável de controle para o laço while. Antes de cada iteração, a condição i < 5 é avaliada. Se a condição for verdadeira, o bloco de código dentro do laço é executado. Se a condição for falsa, o laço termina e a execução continua após o bloco de código do laço.

Laço do-while

A estrutura **do-while** é semelhante à while, mas com uma diferença importante: ela garante que o bloco de código seja executado pelo menos uma vez, mesmo que a condição seja falsa desde o início. Isso ocorre porque a condição só é verificada após a primeira execução do bloco.

Sintaxe:

```
do {
   // Bloco de código a ser repetido
} while (condição)
```

Como funciona ?

O bloco de código é executado uma vez antes da avaliação da condição. Após a execução do bloco de código, a condição é avaliada. Se for verdadeira, o bloco de código é executado novamente.

Exemplo:

```
int i = 0;

do {
   System.out.println(“Contagem ” + i);
   i++;
} while (i < 5);
```

Declaramos e inicializamos a variável i com o valor 0. Essa variável será usada como a variável de controle para o laço do-while.
Após a execução do bloco de código, a condição i < 5 é avaliada. Se a condição for verdadeira, o laço repete a execução do bloco de código. Se a condição for falsa, o laço termina e a execução continua após o bloco de código do laço.

Diferença entre while e do-while

No laço **while**, a condição é verificada antes da execução do bloco de código. Se a condição for falsa no início, o bloco de código pode nunca ser executado.

No laço **do-while**, o bloco de código é sempre executado pelo menos uma vez antes da condição ser verificada. Isso garante que o bloco de código será executado pelo menos uma vez, independentemente da condição inicial.

No exemplo acima, se a variável i fosse inicializada com o valor 5, no laço while nada seria impresso na tela, pois a condição é verificada primeiro. Com i < 5 sendo falso, o bloco de código não seria executado.
Já no laço do-while, o valor 5 seria impresso, pois o bloco de código é executado primeiro e só depois a condição i < 5 é verificada. Portanto, mesmo com i < 5 sendo falso, o bloco de código seria executado pelo menos uma vez.

Conslusão:

Embora as três estruturas de repetição (while, do-while, e for) sejam semelhantes, elas têm seus próprios casos de uso ideais:

- **While:** Use quando não souber de antemão quantas vezes o loop precisará ser executado e quiser garantir que o loop só ocorra enquanto a condição for verdadeira.
- **Do-While:** Use quando quiser garantir que o bloco de código seja executado pelo menos uma vez, independentemente da condição.
- **For:** Use quando souber o número exato de iterações com antecedência.

Palavras-chave de Controle

As palavras-chave de controle break e continue são usadas para manipular o fluxo de execução dentro das estruturas de repetição, oferecendo maneiras mais específicas de controlar quando sair ou pular iterações em loops. Elas são ferramentas poderosas, que permitem interromper loops ou modificar seu comportamento de acordo com condições específicas.

Palavra-chave break

A palavra-chave **break** é usada para interromper um loop imediatamente, saindo do bloco de repetição, mesmo que a condição do loop não tenha sido totalmente satisfeita. Isso é útil em casos onde você deseja encerrar a execução de um loop com base em uma condição específica antes do loop ser concluído normalmente.

Uso do break:

- Interrompe a execução do loop em qualquer ponto e passa a execução para a linha de código logo após o loop.

.

- Funciona em todos os tipos de loops (for, while, do-while), além de ser usado em estruturas de decisão como switch.

Sintaxe Geral:

```
break;
```

Exemplo 1 - Uso em Estrutura for

```
for (int i = 0; i < 10; i++) {
   if (i == 5) {
      break; // O loop será interrompido quando i for igual a 5
   }
   System.out.println("Valor de i: " + i);
}
```

Nesse exemplo, o loop para quando i é igual a 5, mesmo que a condição do for (que é i < 10) ainda seja verdadeira. Assim, as iterações após i = 4 não são executadas.

Exemplo 2 - Uso em Estrutura while

```
int i = 0;
while (i < 10) {
   if (i == 6) {
      break; // Interrompe o loop quando i for igual a 6
   }
   System.out.println("Contador: " + i);
   i++;
}
```

Aqui, o loop while é interrompido assim que o valor de i atinge 6, mesmo que a condição do loop (contador < 10) ainda pudesse continuar.

Palavra-chave continue

A palavra-chave **continue** é usada para pular a iteração atual de um loop e passar diretamente para a próxima iteração. Diferente do break, que interrompe completamente o loop, o continue ignora o restante do bloco de código na iteração em que é chamado e recomeça o loop a partir da próxima iteração, respeitando a condição do loop.

Uso do continue:

- Evita a execução de parte do código em uma iteração específica e segue para a próxima.
- Usada em loops (for, while, do-while) para passar por cima de certos valores ou condições.

Sintaxe Geral:

```
continue;
```

Exemplo 1 - Uso em Estrutura for

```
for (int i = 0; i < 10; i++) {
   if (i == 5) {
      continue; // Pula a iteração quando i for igual a 5
   }
   System.out.println("Valor de i: " + i);
}
```

Nesse exemplo, quando i é igual a 2, o comando continue faz com que o bloco de código restante na iteração atual seja ignorado. A execução continua na próxima iteração (i = 3), sem imprimir o valor 2.

Exemplo 2 - Uso em Estrutura while

```
int i = 0;
while (i < 10) {
   if (i == 6) {
      continue; // Pula a iteração quando i for igual a 6
   }
   System.out.println("Contador: " + i);
   i++;
}
```

Aqui, quando o contador é igual a 3, o comando continue ignora a impressão desse valor, e o loop continua a partir da próxima iteração, com contador igual a 4.

Conslusão:

As palavras-chave break e continue são ferramentas essenciais para controlar de forma mais detalhada o comportamento dos loops em Java. Elas oferecem flexibilidade para sair de loops ou ignorar iterações específicas com base em condições dinâmicas, permitindo que os programas sejam mais eficientes e tenham um controle de fluxo mais sofisticado. Saber quando e como usá-las é uma habilidade importante para evitar loops infinitos e para otimizar a execução de seus programas.

Exercícios

1. Explique a diferença entre uma estrutura de decisão simples (if) e uma estrutura de decisão composta (if-else). Dê exemplos de quando usar cada uma.

2. O que é uma estrutura de repetição? Explique as principais diferenças entre os laços for, while e do-while.

3. Qual é a função das palavras-chave break e continue? Explique como elas afetam o fluxo de controle em loops.

4. O que é uma expressão condicional? Como ela se relaciona com o uso de operadores lógicos em estruturas de controle?

5. Escreva um programa que verifique se um número fornecido pelo usuário é positivo, negativo ou zero.

6. Desenvolva um programa que receba três números do usuário e determine qual deles é o maior.

7. Faça um programa que receba o valor de três lados de um triângulo e determine se o triângulo é equilátero, isósceles ou escaleno.

8. Desenvolva um programa que receba a temperatura de um ambiente e determine se está "frio" (abaixo de 15°C), "agradável" (entre 15°C e 30°C) ou "quente" (acima de 30°C).

9. Faça um programa que receba uma nota de 0 a 10 e exiba a classificação correspondente: A (nota entre 9 e 10), B (nota entre 7 e 8.9), C (nota entre 5 e 6.9), D (nota abaixo de 5). Use switch-case com intervalos.

10. Faça um programa que imprima todos os números de 1 a 10 usando a estrutura de repetição for.

11. Crie um programa que receba um número e imprima todos os números de 0 até esse número usando while.

12. Faça um programa que peça ao usuário uma sequência de números (a sequência termina quando o número 0 é digitado) e calcule a média desses números. Use do-while.

13. Desenvolva um programa que calcule o fatorial de um número fornecido pelo usuário usando a estrutura while.

14. Escreva um programa que exiba todos os números de 1 a 20, mas interrompa o loop quando encontrar o número 13. Use break.

15. Crie um programa que peça ao usuário um número entre 1 e 10. O programa deve imprimir os números de 1 a 10, mas deve pular o número fornecido pelo usuário (usando continue).

16. Desenvolva um programa que imprima a tabuada de multiplicação de um número informado pelo usuário, de 1 a 10. Use for.