Java Developer

Java Core

Sumário

Objetivos deste Módulo

Ao final deste módulo, objetiva-se que o aluno adquira:

Conhecimentos:

- switch, break, continue, return
- Estrutura de repetição for

1. Switch

O comando **switch** em Java é uma estrutura de controle que permite tomar decisões com base no valor de uma expressão. Ele é frequentemente usado como uma alternativa mais limpa e eficiente do que uma sequência de instruções **if-else** aninhadas quando se deseja comparar um valor com vários casos diferentes e executar diferentes blocos de código com base nesses casos.

A estrutura básica de um comando switch em Java é a seguinte:

```
switch (expressao) {
   case valor1:
      // bloco de código a ser executado se expressao for igual a valor1
      break;
   case valor2:
      // bloco de código a ser executado se expressao for igual a valor2
      break;
   // mais casos podem ser adicionados aqui
   default:
      // bloco de código a ser executado se nenhum dos casos anteriores for correspondido
}
```

Aqui estão alguns pontos-chave a serem observados sobre o comando **switch**:

Cada **case** define um valor específico que a expressão será comparada. Se a expressão for igual a um case, o bloco de código correspondente será executado.

Cada bloco de código **case** deve ser finalizado com a instrução **break** para sair do comando switch após o bloco ser executado. Se o break for omitido, a execução continuará nos casos subsequentes, mesmo que a expressão não corresponda a eles. Isso é conhecido como "*fall-through*".

O bloco de código **default** é opcional e será executado quando nenhum dos casos correspondentes for encontrado.

É possível ter **múltiplos case** que compartilham o mesmo bloco de código. Por exemplo:

```
switch (diaDaSemana) {
   case 1:
   case 7:
      System.out.println("Fim de semana");
      break;
   default:
      System.out.println("Dia útil");
}
```

2. Estrutura de repetição: for

O laço **for** em Java é uma estrutura de controle que permite executar um bloco de código repetidamente por um número específico de iterações. Ele é especialmente útil quando você sabe exatamente quantas vezes deseja que o bloco de código seja repetido. A sintaxe básica do **for** em Java é a seguinte:

```java
for (inicialização; condição; atualização) {
    // bloco de código a ser executado em cada iteração
}
```

Aqui estão os componentes do **for**:

Inicialização: Isso é onde você inicializa uma variável de controle ou define um valor inicial. A inicialização ocorre apenas uma vez, no início do loop.

Condição: A condição é uma expressão booleana que determina se o loop deve continuar executando. Enquanto a condição for avaliada como verdadeira, o bloco de código dentro do for será executado.

Atualização: A atualização é responsável por modificar a variável de controle ou qualquer outra coisa necessária para a próxima iteração. Ela é executada após cada iteração do loop.

Aqui está um exemplo simples de um loop **for**:

```java
for (int i = 0; i < 5; i++) {
    System.out.println("Iteração " + i);
}
```

Neste exemplo:

- A variável i é inicializada com 0.
- A condição i < 5 é verificada a cada iteração. Enquanto for verdadeira, o bloco de código será executado.
- Após cada iteração, i é incrementada com 1.

O resultado será a impressão das strings "Iteração 0", "Iteração 1", ..., até "Iteração 4".

O **for** é uma ferramenta poderosa para controlar repetições com um número específico de iterações. Ele é frequentemente usado quando se tem uma contagem conhecida ou quando se precisa percorrer elementos em uma coleção, como arrays ou listas.

Contagem Decrescente:

O **for** também pode ser usado para fazer uma contagem decrescente. Basta ajustar a inicialização, a condição e a atualização de acordo. Por exemplo:

```java
for (int i = 10; i > 0; i--) {
    System.out.println("Contagem decrescente: " + i);
}
```

Nesse caso, o loop imprimirá os números de 10 a 1 em ordem decrescente.

Loop Infinito:

Um **for** pode se tornar um loop infinito se a condição nunca for falsa. Certifique-se de que a condição eventualmente seja avaliada como falsa para que o loop possa terminar. Se você esquecer de atualizar a variável de controle dentro do loop, também poderá causar um loop infinito.

```java
for (int i = 0; i < 5; ) {
    System.out.println(i);
}
```

Neste exemplo, o loop se tornaria infinito porque i nunca é incrementado, e a condição nunca se tornaria falsa.

Uso de Variáveis Externas:

As variáveis usadas na inicialização, condição e atualização não precisam ser necessariamente declaradas dentro do loop **for**. Você também pode usar variáveis já existentes.

```java
int contador = 0;
for (; contador < 3; contador++) {
    System.out.println("Contador: " + contador);
}
```

For-Each Loop (Enhanced for Loop):

O Java também introduziu o loop **for-each**, que é uma forma simplificada de percorrer elementos em uma coleção, como arrays e listas.

```
int[] numeros = { 1, 2, 3, 4, 5 };

for (int numero : numeros) {
    System.out.println(numero);
}
```

Neste exemplo, o **loop for-each** percorrerá cada elemento do array **numeros** e imprimirá o valor de cada elemento.

O loop **for** é uma das estruturas mais versáteis e amplamente usadas em programação, permitindo a iteração controlada e repetitiva através de blocos de código. Ao usar o **for**, certifique-se de que a inicialização, a condição e a atualização estão bem definidas para evitar comportamentos inesperados.

Aninhamento de Loops **for**:

Você pode aninhar **loops** for dentro de outros loops for para criar iterações mais complexas. Isso é útil quando você precisa de múltiplas dimensões de iteração, como em matrizes.

```
int[][] matriz = new int[2][2];

        matriz[0][0] = 10;
        matriz[0][1] = 20;
        matriz[1][0] = 30;
        matriz[1][1] = 40;

        for (int i = 0; i <= 1; i++) {
            for (int j = 0; j <= 1; j++) {
                System.out.println("i: " + i + ", j: " + j);
                System.out.println(matriz[i][j]);
            }
        }
```

Saída no console:

```
i: 0, j: 0
10
i: 0, j: 1
20
i: 1, j: 0
30
i: 1, j: 1
40
```

3. Uso de continue, break e return

Assim como em outros loops, você pode usar as instruções **break** e **continue** em um loop **for** para controlar o fluxo de execução. O **break** interrompe o loop, enquanto o **continue** pula para a próxima iteração.

```
for (int i = 1; i <= 5; i++) {

    if (i == 3) {
        continue; // Pula a iteração quando i é igual a 3
    }

    System.out.println(i);
}

System.out.println("Saiu do loop com continue");
```

Saída no console

```
1
2
4
5
Saiu do loop com continue
```

==========

```
for (int i = 1; i <= 5; i++) {

    if (i == 4) {
        break; // Sai do loop quando i é igual a 4
    }

    System.out.println(i);
}

System.out.println("Saiu do loop com break");
```

Saída no console

```
1
2
3
Saiu do loop com break
```

==========

```
        for (int i = 1; i <= 5; i++) {

            if (i == 3) {
                return;
            }

            System.out.println(i);
        }

        System.out.println("Não vai executar isso pq usou return");
```

Saída no console
1
2

==========

Outro exemplo usando array

```
    public static boolean procurarValor(int[] array, int valor){
        for (int i = 0; i <= array.length -1; i++) {

            System.out.println(array[i]);

            if (array[i] == valor) {
                return true;
            }
        }

        return false;
    }

        int[] numeros = {1,2,3,4,5};
        System.out.println(procurarValor(numeros, 6));
```

Saída no console
1
2
3
4
5
false

Qual a diferença entre continue, break e return em um loop em JAVA?

continue, **break** e **return** são instruções diferentes que têm propósitos distintos em um loop em Java. Vou explicar a diferença entre essas três instruções:

continue:

- O continue é usado para pular a iteração atual do loop e ir para a próxima iteração.
- Ele não encerra o loop; apenas pula para a próxima iteração sem executar o restante do código dentro do loop para essa iteração.
- É útil quando você deseja ignorar parte do código para certas condições e continuar a iteração.
- É frequentemente usado para evitar a execução de código desnecessário e economizar tempo.

break:

- O break é usado para interromper completamente a execução do loop e sair do loop.
- Ele é usado para encerrar o loop antes que a condição normal de término seja alcançada.
- É útil quando você alcança uma condição que exige a saída imediata do loop, independentemente das iterações restantes.
- O break pode ser usado para evitar loops infinitos ou para parar uma busca quando o valor desejado é encontrado.

return:

- O return é usado para sair de um método e, opcionalmente, retornar um valor.
- Se você usar return dentro de um loop, o loop também será encerrado imediatamente, mesmo que ainda haja iterações restantes.
- return é usado quando você deseja encerrar a execução do método completamente, não apenas o loop.

Em resumo:

- **continue** é usado para pular a iteração atual e ir para a próxima iteração no loop.
- **break** é usado para encerrar completamente o loop e sair dele.
- **return** é usado para encerrar a execução do método atual e, opcionalmente, retornar um valor.

A escolha entre essas instruções depende do comportamento desejado no contexto do seu código. Cada uma tem um propósito específico e é usada para controlar o fluxo do programa de maneira diferente.

4. Exercícios

1) Escreva um programa que use um loop **for** para imprimir os números pares de 1 a 20, **pulando** as iterações ímpares.
2) Escreva um programa que calcule a soma dos elementos em um **array** de números inteiros.
3) Escreva um programa que encontre o maior e o menor elemento em um **array** de números inteiros.
4) Crie um programa que atue como um conversor de unidades. O usuário poderá escolher entre converter temperatura de Celsius para Fahrenheit ou converter distância de quilômetros para milhas. Utilize o comando **switch** para lidar com as diferentes opções.

```
System.out.println("Conversor de Unidades:");
System.out.println("1. Converter Celsius para Fahrenheit");
System.out.println("2. Converter Quilômetros para Milhas");
```

Fórmula para converter Celsius para Fahrenheit: (celsius * 9 / 5) + 32
Fórmula para converter Km em milhas: km * 0.621371: