Java Developer

Java Core

Sumário

Objetivos deste Módulo

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

Conhecimentos:

- Controle de fluxo: if, else if, else
- Operadores aritméticos, incremento, decremento, atribuição com operações aritméticas, unários, relacionais e lógicos
- Arrays

1. Controle de fluxo if, else if, else

O controle de fluxo em Java refere-se à capacidade de alterar o fluxo de execução do programa, permitindo que diferentes trechos de código sejam executados em diferentes condições. Isso é feito usando estruturas de controle, como **condicionais** (if, else if, else) e **loops** (for, while, do-while). Veremos nesta aula as estruturas condicionais e futuramente os loops (também chamamos de estruturas de repetição).

Estruturas Condicionais:

As estruturas condicionais permitem que você execute diferentes trechos de código com base em condições específicas. A principal estrutura condicional em Java é o "if".

- **if**: O bloco de código dentro do "if" é executado se a condição especificada for verdadeira.

```java
if (condicao) {
    // código a ser executado se a condição for verdadeira
}
```

- **else if**: Permite verificar condições adicionais se a condição do "if" não for verdadeira.

```java
if (condicao1) {
    // código a ser executado se a condicao1 for verdadeira
} else if (condicao2) {
    // código a ser executado se a condicao2 for verdadeira
}
```

- **else**: O bloco de código dentro do "else" é executado se nenhuma das condições anteriores for verdadeira.

```java
if (condicao) {
    // código a ser executado se a condição for verdadeira
} else {
    // código a ser executado se a condição não for verdadeira
}
```

Vejamos um exemplo:

```java
import java.util.Scanner;

public class ExemploControleFluxo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int numero = scanner.nextInt();

        if (numero > 0) {
            System.out.println("O número é positivo.");
        } else if (numero < 0) {
            System.out.println("O número é negativo.");
        } else {
            System.out.println("O número é zero.");
        }

        scanner.close();
    }
}
```

Neste exemplo, o programa solicita ao usuário para digitar um número. Com base no valor do número, o programa usa as estruturas condicionais para determinar se o número é positivo, negativo ou zero, e em seguida, imprime a mensagem apropriada. Se o número for maior que zero, o bloco de código dentro do primeiro "if" será executado. Se o número for menor que zero, o bloco dentro do "else if" será executado. Caso contrário, o bloco dentro do "else" será executado. Isso demonstra como o controle de fluxo permite que diferentes partes do código sejam executadas com base nas condições especificadas.

2. Operadores aritméticos

Em Java, os operadores aritméticos são usados para realizar operações matemáticas em valores numéricos. Eles permitem que você realize adições, subtrações, multiplicações, divisões e outras operações matemáticas em variáveis ou constantes numéricas. Aqui estão os operadores aritméticos em Java:

- **Adição (+)**: Realiza a adição de dois valores.

```java
int resultado = valor1 + valor2;
```

- **Subtração (-)**: Realiza a subtração entre dois valores.

```java
int resultado = valor1 - valor2;
```

- **Multiplicação (*)**: Realiza a multiplicação de dois valores.

```java
int resultado = valor1 * valor2;
```

- **Divisão (/)**: Realiza a divisão de um valor pelo outro. Se os operandos forem inteiros, o resultado será um número inteiro.

```java
double resultado = valor1 / valor2;
```

- **Módulo (%)**: Retorna o resto da divisão entre dois valores.

```java
int resto = valor1 % valor2;
```

É importante lembrar que a ordem de avaliação dos operadores segue as regras matemáticas padrão. Você também pode usar parênteses para definir a precedência de operações, assim como faria em expressões matemáticas.

Exemplo:

```java
int resultado = (valor1 + valor2) * valor3;
```

Teste a diferença com o código abaixo:

System.out.println(3+3*3); // resultado é 12
System.out.println((3+3)*3); // resultado é 18

3. Operadores de incremento e decremento

- **Incremento (++)**: Incrementa o valor de uma variável em 1. Pode ser usado antes ou depois da variável. O código abaixo é o mesmo que fazer `valor = valor + 1`

```java
int valor = 5;
valor++; // valor agora é 6
```

- **Decremento (--)**: Decrementa o valor de uma variável em 1. Pode ser usado antes ou depois da variável. O código abaixo é o mesmo que fazer `valor = valor - 1`

```java
int valor = 5;
valor--; // valor agora é 4
```

4. Operadores de atribuição com operações aritméticas

- Esses operadores combinam uma operação aritmética com uma atribuição. Eles são uma forma concisa de atualizar o valor de uma variável baseado em uma operação. São eles:
  - +=
  - -=
  - /=
  - %=

```java
int valor = 10;
valor += 5; // Equivalente a: valor = valor + 5;
```

5. Operadores unários

- Esses operadores são usados para indicar o sinal de um valor numérico. O operador "+" é opcional, pois os números positivos são representados sem ele. O operador "-" negativa o valor.

```java
int positivo = +10; // Pode ser omitido: int positivo = 10;
int negativo = -10;
```

6. Operadores relacionais

- Maior que (>): Verifica se o valor à esquerda é maior que o valor à direita.
- Menor que (<): Verifica se o valor à esquerda é menor que o valor à direita.
- Maior ou igual (>=): Verifica se o valor à esquerda é maior ou igual ao valor à direita.
- Menor ou igual (<=): Verifica se o valor à esquerda é menor ou igual ao valor à direita.
- Igual (==): Verifica se os valores à esquerda e à direita são iguais.
- Diferente (!=): Verifica se os valores à esquerda e à direita são diferentes.

```java
int valor1 = 10;
int valor2 = 20;
boolean resultado = valor1 > valor2; // falso
```

Lembre-se de que, ao usar operadores aritméticos, você precisa levar em consideração os tipos de dados das variáveis envolvidas para garantir que as operações sejam realizadas corretamente. Tipos incompatíveis podem resultar em erros ou em resultados inesperados. Além disso, a ordem de operações pode ser controlada usando parênteses para garantir a avaliação correta.

7. Operadores lógicos

Operadores lógicos em Java são usados para realizar **operações de lógica booleana**, ou seja, para avaliar e combinar expressões lógicas que resultam em valores verdadeiros (true) ou falsos (false). Esses operadores são fundamentais para criar condições e decisões em programas, permitindo que você avalie múltiplas expressões lógicas e determine o fluxo de execução com base nessas avaliações.

Aqui estão os principais operadores lógicos em Java:

- **AND (&&)**: Retorna verdadeiro se **ambas** as expressões forem **verdadeiras**.

```java
boolean resultado = expressao1 && expressao2;
```

- **OU (||)**: Retorna **verdadeiro** se **pelo menos uma** das expressões for verdadeira.

```java
boolean resultado = expressao1 || expressao2;
```

- **NÃO / NEGAÇÃO (!)**: **Inverte** o **valor** de uma expressão, ou seja, se a expressão for verdadeira, o resultado será falso e vice-versa.

```java
boolean resultado = !expressao;
```

Esses operadores lógicos permitem que você crie condições mais complexas ao combinar expressões lógicas simples. Eles são frequentemente usados em instruções condicionais (if, else if, else) e em estruturas de repetição (loops) para controlar o fluxo de execução do programa com base em condições verdadeiras ou falsas.

Exemplo:

```
import java.util.Scanner;

public class ExemploOperadoresLogicos {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a sua idade: ");
        int idade = scanner.nextInt();

        System.out.print("Você tem carteira de motorista? (true/false): ");
        boolean temCarteira = scanner.nextBoolean();

        // Verifica se a pessoa é elegível para dirigir
        if (idade >= 18 && temCarteira) {
            System.out.println("Você pode dirigir.");
        } else {
            System.out.println("Você não pode dirigir.");
        }

        scanner.close();
    }
}
```

Neste exemplo, o programa solicita ao usuário para digitar sua idade e se possui carteira de motorista. O programa usa o operador lógico && (E lógico) para verificar se ambas as condições são verdadeiras: idade

maior ou igual a 18 anos e ter carteira de motorista. Se ambas as condições forem verdadeiras, a pessoa é considerada elegível para dirigir e o programa imprime a mensagem "Você pode dirigir". Caso contrário, imprime a mensagem "Você não pode dirigir".

Operadores lógicos são uma parte crucial da programação para tomar decisões com base em condições e criar lógica complexa em seus programas Java.

8. Arrays

Em Java, um array é uma estrutura de dados que permite armazenar um conjunto de valores do mesmo tipo sob um único nome. Os elementos em um array são organizados de forma sequencial e podem ser acessados por meio de um índice numérico. Os arrays são úteis para armazenar coleções de dados relacionados, como números, caracteres, objetos, etc.

Aqui estão os principais conceitos relacionados a arrays em Java:

- **Declaração e Criação de Arrays**: Você declara um array especificando o tipo de dados que ele armazenará e o nome do array. Em seguida, você cria o array utilizando o operador **new** e especificando o tamanho desejado.

java Copy code

```java
// Declaração e criação de um array de inteiros com tamanho 5
int[] numeros = new int[5];
```

```java
int[] numeros = new int[5];
numeros[0] = 100;
System.out.println(numeros[0]); // exibe 100
```

- **Inicialização de Arrays**: Você pode inicializar os elementos de um array no momento da criação ou posteriormente. Os valores iniciais são definidos entre chaves {}.

java Copy code

```java
// Inicialização de um array de inteiros
int[] numeros = {1, 2, 3, 4, 5};
```

E para exibir o array:

```java
int[] numeros = {1, 2 , 3 , 4, 5 };
System.out.println(numeros);// imprime uma referência de memória
System.out.println(Arrays.toString(numeros)); // [1, 2, 3, 4, 5]
```

- **Acesso aos Elementos**: Os elementos de um array são acessados usando um índice baseado em zero. O primeiro elemento está no índice 0, o segundo no índice 1 e assim por diante.

```java
int primeiroElemento = numeros[0]; // Acessa o primeiro elemento
int segundoElemento = numeros[1];  // Acessa o segundo elemento
```

- **Tamanho do Array**: Você pode obter o tamanho de um array usando a propriedade length.

```java
int tamanho = numeros.length; // Retorna o tamanho do array
```

- **Arrays Multidimensionais**: Java permite criar arrays multidimensionais, como matrizes (arrays de duas dimensões).

```java
int[][] matriz = new int[3][3]; // Matriz 3×3
```

```java
int[][] matriz = new int[3][3]; // Matriz 3x3
matriz[0][0] = 15; // primeira linha, primeira coluna
System.out.println(matriz[0][0]); // exibe 15
```

9. Exercícios

1) Faça um método que recebe duas notas e retorne a maior entre elas.

2) Solicite ao usuário que informe um número. Se o valor for par, informe “é par”, senão “é ímpar”

3) Implemente o algoritmo do IMC

   FÓRMULA DO IMC = Peso ÷ (Altura × Altura)

   O usuário deve informar peso e altura. E o sistema deverá exibir a classificação após calcular a fórmula do IMC.

IMC	CLASSIFICAÇÃO	OBESIDADE (GRAU)
Menor que 18,5	Magreza	0
Entre 18,5 e 24,9	Normal	0
Entre 25,0 e 29,9	Sobrepeso	I
Entre 30,0 e 39,9	Obesidade	II
Maior que 40,0	Obesidade Grave	III

Dica: se quiser formatar o resultado final do imc com duas casas decimais utilize o método format da classe String

```
String resultado = String.format("%.2f", 123.9633584);
System.out.println(resultado); // exibe 123,96
```

4) Solicite ao usuário que digite 3 notas e armazene-as em um array. Some as 3 posições do array e calcule a média aritmética. Caso a nota seja >= 7, exiba aprovado, caso contrário, reprovado.