



Quem se prepara, não para.



# Linguagens de Programação

3º período

Prof. Dr. João Paulo Aramuni

# Sumário

- Java
  - Estruturas de repetição
    - Laço while e do-while
    - Break e Continue
  - Uso de valores inteiros e reais
  - Formatação da saída
  - Módulos

# Sumário

- Java
  - Estruturas de repetição
    - **Laço while e do-while**
    - Break e Continue
  - Uso de valores inteiros e reais
  - Formatação da saída
  - Módulos

- Estruturas de repetição

- Laço while

- Formatação geral:

```
while(condição) instrução;
```

- Onde *instrução* pode ser uma única instrução ou um bloco de instruções, e *condição* define a condição que controla o laço. A condição pode ser qualquer expressão booleana válida.
      - O laço se repete enquanto a condição é verdadeira. Quando a condição se torna falsa, o controle do programa passa para a linha imediatamente posterior ao laço.

- Estruturas de repetição
  - Laço while

```
1 package packageteste.modulo;
2
3 public class Teste {
4     public static void main(String[] args) {
5
6         // Demonstra o laço while
7         char ch;
8         // exibe o alfabeto usando um laço while
9         ch = 'a';
10        while(ch <= 'z') {
11            System.out.print(ch);
12            ch++;
13        }
14
15    }
16 }
```

Console × Problems @ Javadoc Declaration Search

<terminated> Teste (1) [Java Application] /Library/Java/JavaVirtualMachin

abcdefghijklmnopqrstuvwxyz

- Estruturas de repetição

- Laço do-while

- Formatação geral:

```
do {  
    instruções;  
} while (condição);
```

- O laço do-while verifica sua condição no fim do laço. Ou seja, um laço do-while será sempre executado **pelo menos uma vez**.
      - Embora as chaves não sejam necessárias quando há apenas uma instrução presente, elas são usadas com frequência para melhorar a legibilidade da estrutura do-while.

- Estruturas de repetição

- Laço do-while

- Formatação geral:

```
do {  
    instruções;  
} while (condição);
```

- O laço do-while é executado enquanto a expressão for verdadeira.
      - É comum utilizar o do-while para a criação de menus ou quaisquer outras regras de negócio que necessitem serem rodadas ao menos uma vez.



- Estruturas de repetição

- Laço do-while

- O do-while realiza a leitura do teclado antes de testar o caractere lido no while.
    - Há também outro do-while aninhado para limpar o buffer de entrada.

```
// Adivinhe a letra do jogo, 4a versão.
char ch, ignore, answer = 'K';
do {
    System.out.println("I'm thinking of a letter between A and Z.");
    System.out.print("Can you guess it: ");

    // lê um caractere
    ch = (char) System.in.read();

    // descarta qualquer outro caractere do buffer de entrada
    do {
        ignore = (char) System.in.read();
    } while (ignore != '\n');

    if (ch == answer)
        System.out.println("** Right **");
    else {
        System.out.print("...Sorry, you're ");

        if (ch < answer)
            System.out.println("too low");
        else
            System.out.println("too high");

        System.out.println("Try again!\n");
    }
} while (answer != ch);
```

# Sumário

- Java
  - Estruturas de repetição
    - Laço while e do-while
    - **Break e Continue**
  - Uso de valores inteiros e reais
  - Formatação da saída
  - Módulos

- **Estruturas de repetição**

- **Break**

- É possível forçar a saída imediata de um laço, ignorando o código restante em seu corpo e o teste condicional, com o uso da instrução break. Quando uma instrução break é encontrada dentro de um laço, este é encerrado e o controle do programa é retomado na instrução posterior ao laço.
    - Tome cuidado: Usar o break de forma excessiva ou desnecessária, torna o código mais difícil de entender e manter. Quando utilizado indiscriminadamente, o break pode levar a códigos complexos e difíceis de depurar, especialmente quando há múltiplos loops aninhados com break em diferentes pontos.

- Estruturas de repetição
  - Break

```
40
41 // Usando break para sair de um laço.
42 int num = 100;
43 // executa o laço enquanto i ao quadrado é menor do que num
44 for (int i = 0; i < num; i++) {
45     if (i * i >= num)
46         break; // encerra o laço se i*i >= 100
47     System.out.print(i + " ");
48 }
49 System.out.println("Loop complete.");
50
51 }
52 }
53
```

Console × Problems @ Javadoc Declaration Search Breakpoints

<terminated> Teste (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (25 c

0 1 2 3 4 5 6 7 8 9 Loop complete.

- Estruturas de repetição
  - Continue
    - É possível forçar uma iteração antecipada de um laço, ignorando sua estrutura de controle normal. Isso é feito com o uso de **continue**. A instrução continue força a ocorrência da próxima iteração do laço e qualquer código existente entre ela e a expressão condicional que controla o laço é ignorado.
    - Logo, **continue** é basicamente o complemento do **break**.

- Estruturas de repetição
  - Continue

```
51 // Usando continue.
52 int i;
53 // exibe os números pares entre 0 e 100
54 for (i = 0; i <= 100; i++) {
55     if ((i % 2) != 0)
56         continue; // iterate
57     System.out.println(i);
58 }
59
60 }
61 }
```

Console × Problems Javadoc Declaration Search Breakpoint

<terminated> Teste (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Con

```
80
82
84
86
88
90
92
94
96
98
100
```

# Sumário

- Java
  - Estruturas de repetição
    - Laço while e do-while
    - Break e Continue
  - **Uso de valores inteiros e reais**
  - Formatação da saída
  - Módulos

- **Uso de valores inteiros e reais**
  - **Valores inteiros**

```
byte b = 127;
```

```
short s = 32767;
```

```
int i = 2147483647;
```

```
long l = 9223372036854775807L;
```

```
// Valores long são representados com L ou l no final,
```

```
// caso contrário são int
```



- Uso de valores inteiros e reais
  - Valores reais

```
float f = 3.4028235E38F;
```

```
double d = 1.7976931348623157E308;
```

$$7.1\text{E}2 = 7.1 \times 10^2$$
$$7.1\text{e}2 = 7.1 \times 10^2$$

```
// Valores float são representados com F ou f no final,
```

```
// caso contrário são double
```

```
// Estes são os limites positivos
```

- **Uso de valores inteiros e reais**

- **float vs double**

- A principal diferença entre float e double é a precisão.
    - O tipo **double** oferece uma precisão maior e é geralmente usado quando a precisão extra é necessária, especialmente em cálculos matemáticos complexos ou em aplicações onde a exatidão dos valores é importante.
    - Por outro lado, **float** é usado quando a precisão extra não é necessária e quando economia de memória é importante, como em arrays grandes de números de ponto flutuante.
    - Geralmente, é recomendado usar **double** na maioria dos casos, a menos que haja uma razão específica para usar **float**.

- **Leitura pelo teclado**
  - Vimos anteriormente como ler um um **char** via teclado:

```
156
157     // Lê um caractere no teclado.
158     char chx;
159     System.out.print("\nPress a key followed by ENTER: ");
160     chx = (char) System.in.read(); // obtém um char
161     System.out.println("Your key is: " + chx);
162
163 }
164 }
165
166
```

Console × Problems @ Javadoc Declaration Search Breakpoints

<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/H

Press a key followed by ENTER: A  
Your key is: A

- **Leitura pelo teclado**

- Vejamos agora como ler uma **string**:
  - **import** java.util.Scanner();
    - Observação:
      - Prefira import java.util.Scanner;
      - ao invés de import java.util.\*;
- A classe **Scanner** em Java faz parte do pacote java.util e fornece uma maneira fácil e eficiente de ler dados de entrada de várias fontes, incluindo o **teclado** (System.in) e **arquivos**.
- É uma das formas mais comuns de interação com o usuário e de ler dados do console.

- Leitura pelo teclado

```
62 // in representa o teclado
63 Scanner in = new Scanner(System.in);
64
65 System.out.println("\nDigite seu nome: ");
66 String nome = in.nextLine();
67 System.out.println("Digite sua idade: ");
68 int idade = in.nextInt();
69 System.out.println("Digite seu salário: ");
70 double salario = in.nextDouble();
71 in.close();
72
73 System.out.println("\n" + nome + "\n" + idade + "\n" + salario);
74
75 }
76 }
```

Console X Problems Javadoc Declaration Search Breakpoints

<terminated> Teste (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (25 de jul. de 2023 10

Digite seu nome:  
Aramuni  
Digite sua idade:  
27  
Digite seu salário:  
30000

Aramuni  
27  
30000.0

# Sumário

- Java
  - Estruturas de repetição
    - Laço while e do-while
    - Break e Continue
  - Uso de valores inteiros e reais
  - **Formatação da saída**
  - Módulos

- **Formatação de saída**
  - A formatação das casas decimais irá utilizar as configurações regionais do computador.

```
// Formatação de saída
double x = 10000.0 / 3.0;

System.out.print(x);           // 3333,333333333335
System.out.printf("%, .3f", x); // 3.333,333
System.out.printf("R$ %, .2f", x); // R$ 3.333,33
```

- **Formatação de saída**

- O método **printf** recebe mais de um argumento, o primeiro sempre é o **formato** (tipo string - indica o texto que será impresso), seguido por **valores**.

```
System.out.printf(formato, valor1, valor2, ...);
```

- Existem vários conversores, os mais utilizados são:

<code>%d</code>	<code>int</code>
<code>%c</code>	<code>char</code>
<code>%s</code>	<code>String</code>
<code>%f</code>	<code>double e float</code>

<code>printf("%.2f", 10.5);</code>	<b>10.50</b>	Formata com 2 casas decimais
<code>printf("%,d", 17435);</code>	<b>17,435</b>	Formata separando na casa dos milhares
<code>printf("%02d", 6);</code>	<b>06</b>	Formata com 2 dígitos, completando com zeros
<code>printf("%+f", 13.7);</code>	<b>+13.700000</b>	Formata forçando a exibição do sinal



# Sumário

- Java
  - Estruturas de repetição
    - Laço while e do-while
    - Break e Continue
  - Uso de valores inteiros e reais
  - Formatação da saída
  - **Módulos**

- **Módulos**

- A partir do Java 9, o Java introduziu o conceito de "**módulos**" (modules) como parte importante do **Jigsaw Project**, também conhecido como Projeto Jigsaw.
- Antes do Projeto Jigsaw, o Java era conhecido por sua natureza **monolítica**, onde todas as bibliotecas e classes do Java SE (Standard Edition) faziam parte do classpath global, tornando difícil gerenciar dependências e garantir a encapsulação do código.
- O Projeto Jigsaw foi criado para abordar essas questões e introduziu o conceito de módulos no Java.

- **Módulos**

- Os principais objetivos do Projeto Jigsaw são:
  - **Modularidade:** Dividir a plataforma Java em módulos bem definidos, permitindo que as aplicações sejam construídas e implantadas em unidades menores e mais coesas.
  - **Encapsulamento:** Definir visibilidade e controle rigorosos entre os módulos, permitindo que cada módulo exponha apenas o que é necessário para outros módulos.
  - **Eficiência:** Melhorar o desempenho e o consumo de memória ao carregar apenas os módulos necessários para uma aplicação específica.
  - **Escalabilidade:** Facilitar o desenvolvimento de aplicativos maiores e complexos, fornecendo uma estrutura mais organizada e gerenciável.

- **Módulos**

- Os módulos são uma forma de encapsular e organizar o código em unidades lógicas e independentes no Java.
  - Um módulo é uma unidade fundamental de encapsulamento no Java que agrupa código relacionado, recursos e dependências em um único pacote.
  - Ele define o que é exportado (visível para outros módulos) e o que é mantido privado (não visível para outros módulos). Isso ajuda a melhorar a legibilidade, manutenção e segurança dos aplicativos Java, permitindo um melhor controle sobre as dependências e evitando a poluição do espaço de nomes.

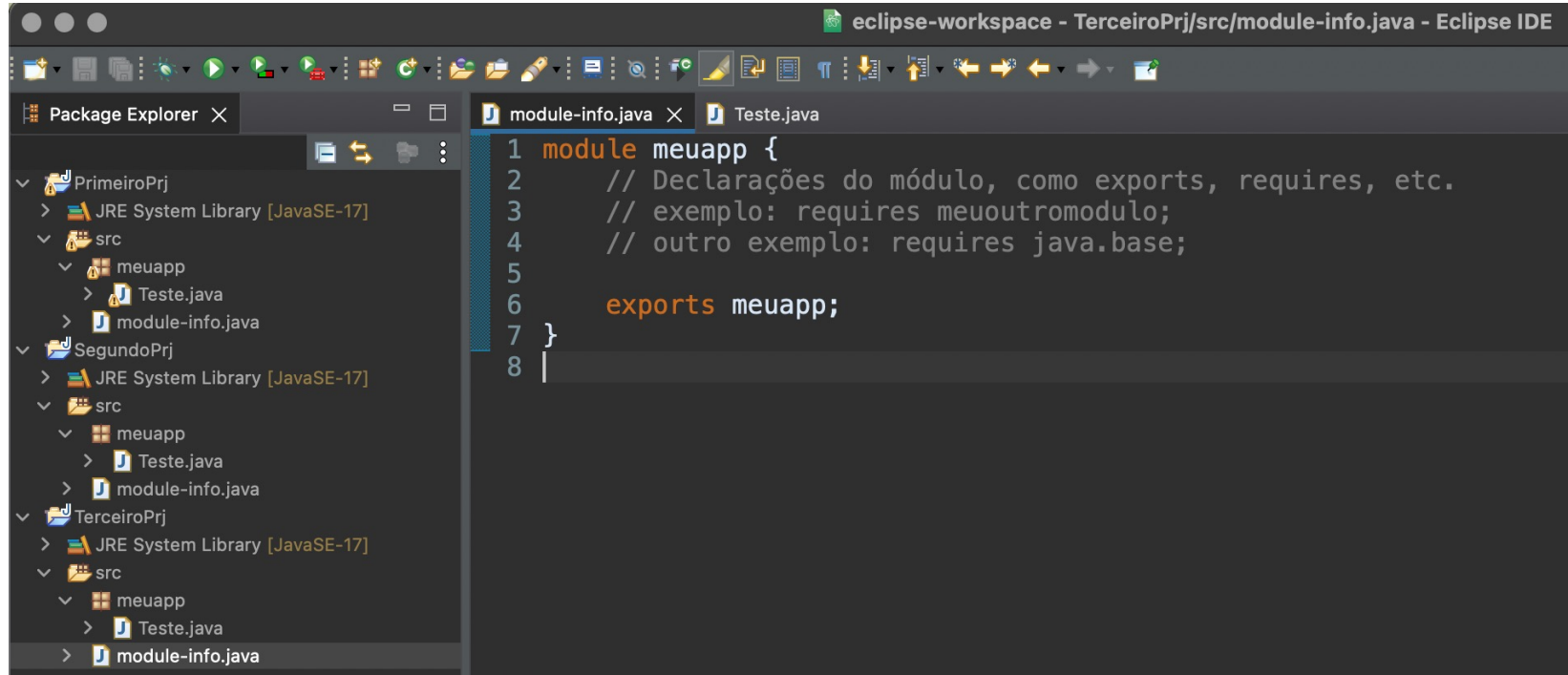
- **Módulos**

- Os módulos no Java 17 e nas versões posteriores continuam a usar o sistema de módulos introduzido no Java 9.
- Com o uso de módulos, os desenvolvedores podem criar aplicativos Java mais eficientes e modulares, permitindo a redução do tamanho do pacote, melhor inicialização e menor consumo de memória.
- Além disso, os módulos são especialmente úteis no contexto do desenvolvimento de aplicativos empresariais complexos, onde a **separação** clara de responsabilidades entre diferentes componentes é crucial para a **escalabilidade** e a **manutenção**.

- **Módulos**

- No Java, você pode declarar um módulo usando o arquivo de manifesto **module-info.java**, que deve estar localizado na **raiz do módulo**.
- Esse arquivo especifica as **dependências**, **pacotes exportados** e outros detalhes relacionados ao módulo.
- No geral, os módulos no Java oferecem uma maneira mais estruturada e eficiente de desenvolver aplicativos, promovendo a modularidade, a organização do código e a clareza nas dependências.

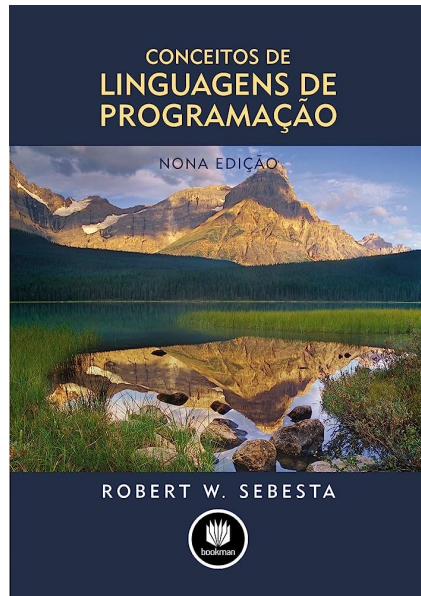
- Módulos



The screenshot shows the Eclipse IDE interface. The title bar reads "eclipse-workspace - TerceiroPrj/src/module-info.java - Eclipse IDE". The Package Explorer on the left shows a project structure with three projects: "PrimeiroPrj", "SegundoPrj", and "TerceiroPrj". Each project contains a "src" folder with "meuapp" and "Teste.java" files. The "TerceiroPrj" project is selected, and its "module-info.java" file is open in the editor. The code in the editor is as follows:

```
1 module meuapp {  
2     // Declarações do módulo, como exports, requires, etc.  
3     // exemplo: requires meuoutromodulo;  
4     // outro exemplo: requires java.base;  
5  
6     exports meuapp;  
7 }  
8 |
```

# Referências







Obrigado!

joaopauloaramuni@gmail.com