Formação Java Developer

BLOCO 1 | CURSO 1 | SEÇÃO 1 | AULA 1 – Introdução e objetivos

Principais aplicações Java de forma básica e inicial

Java fundamentos – Base para fundamentos para a linguagem

Apresentar a história e configurar o ambiente para uso Java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BLOCO 1 | CURSO 1 | SEÇÃO 1 | AULA 2 – O que é Java

A Linguagem Java é composta por um conjunto bem definido de funcionalidades (features).

Funcionalidades essenciais vão ser o foco inicial.

É uma linguagem simples e fácil de entender. É orientada a objetos. Significa que em Java tudo é escrito em termos de classes e Objetos.

POO – Programação Orientada a Objetos.

1 – Classe e Objeto

2 – Encapsulamento

3 – Abstração

4 – Herança

5 – Polimorfismo

Os programas Java são compilados para gerar arquivos de bytecode e a JVM (Java Virtual Machine) interpreta o arquivo de bytecode durante a execução.

BLOCO 1 | CURSO 1 | SEÇÃO 2 | AULA 1 – Java e sua História

Cafeteria em uma cidade chamada Java. Crescente e inovadora em 1993 aproveitando o crescimento da WWW.

Ambientes Java:

Páginas web | Aplicações corporativas | Softwares destinados ao consumidor final | Aplicativos para smartphones.

BLOCO 1 | CURSO 2 | SEÇÃO 1 | AULA 1 – Introdução a IDEs

Apresentação de Interfaces de Desenvolvimento Integrados.

- Eclipse

- IntelliJ

- Visual Studio Code

- NetBeans

IDE usada por mim VSCode.

BLOCO 1 | CURSO 2 | SEÇÃO 1 | AULA 2 – Configuração do Ambiente

Para começar a desenvolver em Java é necessário que tenhamos o JDK instalado.

O **JDK** (Java Development Kit) é um conjunto de ferramentas indispensáveis para o desenvolvimento de aplicações em **Java**. Ele é um ambiente de desenvolvimento que fornece todos os recursos necessários para escrever, compilar e executar programas Java.

**Principais componentes do JDK:**

1. **Compilador (javac)**: Transforma o código-fonte Java (arquivos .java) em bytecode (arquivos .class), que é interpretado pela JVM.
2. **Java Runtime Environment (JRE)**: Inclui a JVM (Java Virtual Machine) e bibliotecas essenciais para executar programas Java. O JRE também é parte do JDK.
3. **Ferramentas de Desenvolvimento**:
   * **Debugger (jdb)**: Ajuda a depurar código.
   * **JavaDoc**: Gera documentação em HTML a partir de comentários no código.
   * **Jar**: Permite criar arquivos compactados .jar para distribuir aplicações Java.
   * Outras ferramentas, como javap (para descompilar bytecode) e java (para executar aplicações Java).

**Uso do JDK**

* **Desenvolvimento**: É usado para criar aplicativos Java, tanto desktop, quanto para web, dispositivos móveis e outras plataformas.
* **Aprendizado**: Ferramenta essencial para quem está aprendendo a programar em Java, pois contém tudo que você precisa para compilar e testar código.

**Download e versões**

A Oracle disponibiliza o JDK gratuitamente no site oficial, mas também há implementações de terceiros, como o OpenJDK (uma versão open-source do JDK).

**Importância**

Sem o JDK, você não conseguiria desenvolver ou compilar programas em Java, pois ele contém o compilador e as ferramentas necessárias para transformar o código em algo executável pela JVM.

**Instalando o JDK**

JDK 23 x64 Installer instalada no meu PC.

Local instalado: C:\Program Files\jdk-23\

Texto

Descrição gerada automaticamente

BLOCO 1 | CURSO 2 | SEÇÃO 2 | AULA 1 – Instalando VSCode

Instalado o Java in Visual Studio Code

Instalado Extensões Java por meio do arquivo baixado (Coding Pack for Java)

Arquivo PrimeiroProjeto criado (Java)

Tela de vídeo game

Descrição gerada automaticamente

Fiz a configuração de tamanho de fonte em Open settings (configurações parte inferior esquerda) User e indo em font size.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Final de aula testei o arquivo App.java e escrevi nele o texto apresentado acima.

BLOCO 1 | CURSO 2 | SEÇÃO 2 | AULA 2 – Instalando outras IDEs

Foi mostrado como instalar outras IDEs para desenvolvimento Java

BLOCO 1 | CURSO 3 | SEÇÃO 1 | AULA 1 – Introdução

Apresentação do curso para informar a aplicação da sintaxe na linguagem Java

Vamos ver nesse módulo:

- Estrutura inicial;

- Padrão de nomenclatura;

- Declarando variáveis e métodos;

- Identação;

- Organizando os arquivos;

- Java Beans.

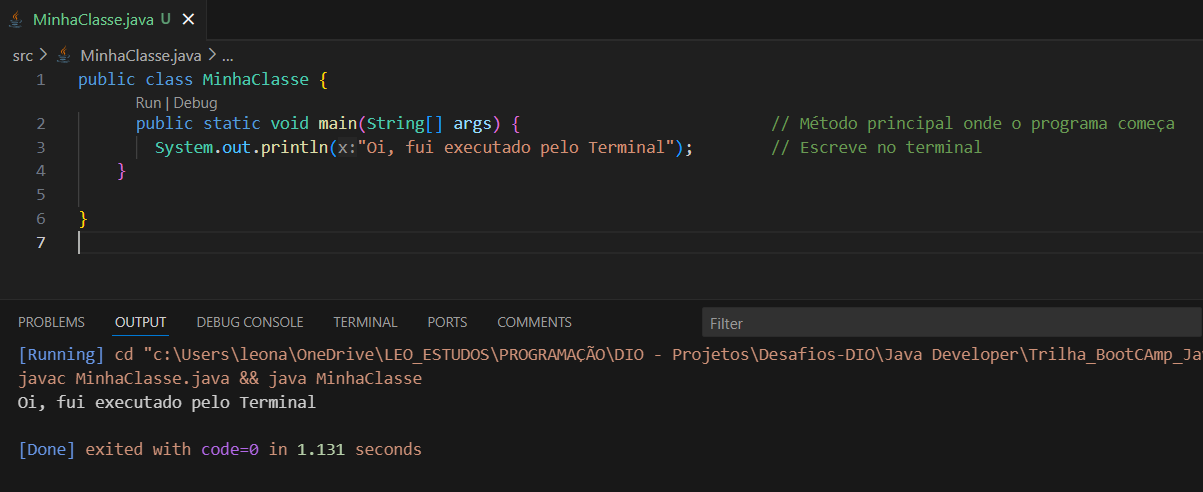
BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 1 – Anatomia das classes 1

Public class NomeDaMinhaClasse {

// SEU CÓDIGO AQUI

}

Para criar um projeto JAVA, no VsCode, basta clicar no campo de pesquisa e clicar em mostrar comandos (show run commands) ou CTRL + SHIFT + P. Escolher Java new Project. Logo após será pedido o diretório onde ficará seu projeto.



Tela de computador

Descrição gerada automaticamente

Criada uma classe (MinhaClasse) para entender cada parte da sintaxe do Java

System.out.print – é o comando para mostrar algo na tela.

BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 2 – Anatomia das classes - 2

Continuando a estrutura do Java

Quando se trata de escrever códigos na linguagem Java, é recomendado seguir algumas convenções de escrita.

- Arquivo.java: Todo arquivo.java deve começar com letra MAIÚSCULA. Se a palavra for composta, a segunda palavra deve também ser maiúscula.

Ex.: Calculadora.java | CalculadoraCientifica.java

O nome da classe deve ser o mesmo nome do arquivo.

- Nome de variável: Toda variável deve ser escrita com letra minúscula, porém se a palavra for composta, a primeira letra da segunda palavra deverá ser maiúscula.

- Para tornar o valor de uma variável imutável, existe uma escrita especial que deixa característica:

Ex.: String br = “Brasil”

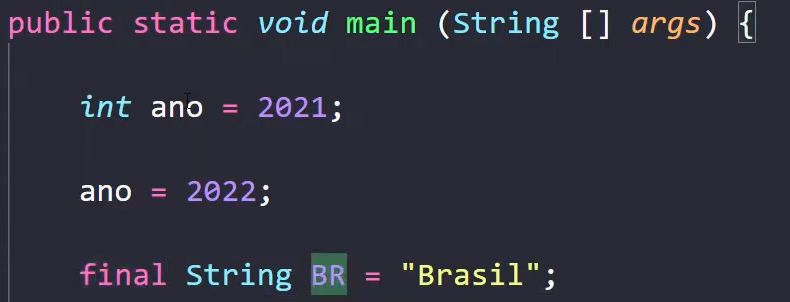
// para essa variável do tipo string a variável br recebeu o valor “Brasil”

E pode sofrer alterações (porque está em minúscula) no valor com o decorrer do tempo de aplicação.

Mas,

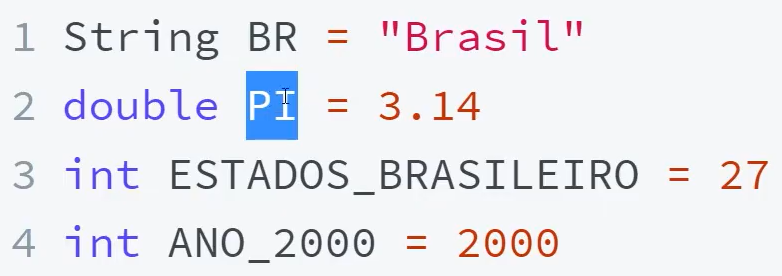
Se escrevermos assim:

Final String BR = “Brasil” – a variável em maiúscula deixa ela sem poder sofrer alterações.



A palavra *final* faz essa função. Deixa a variável sem poder sofrer alterações.

Ou seja, usamos final e colocamos a variável em maiúscula



Exemplo de variáveis que não podem sofrer alterações.

Boas práticas para escrita de variáveis:

- Deve conter apenas letras, underline (\_), sifão ($), ou números de 0 a 9;

- Deve obrigatoriamente se iniciar por uma letra (preferencialmente), \_ ou $, jamais com número;

- Deve iniciar com letra minúscula;

- Não pode conter espaços;

- Não usar palavras-chave da linguagem;

- O nome deve ser único dentro de um escopo;

BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 3 – Anatomia das classes - 3

Declarando variáveis e métodos

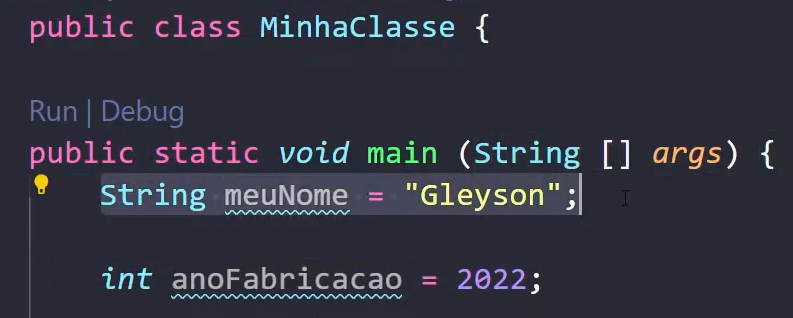
Como identificar entre declaração de variável e métodos em nosso programa? Existe uma estrutura comum para ambas as finalidades.

Declarar uma variável em Java segue sempre a seguinte estrutura:

Tipo NomeBemDefinido = Atribuição

Int idade = 23

Double altura = 1.62



Declarando métodos (funções) em Java segue uma estrutura bem simples:

TipoRetorno NomeObjetivoNoInfinitivo Parametros(s)

Int somar (int numeroUm, int numero2)

String formatarCep (long cep)

Public static String nomeCompleto (String primeiroNome, String segundoNome)

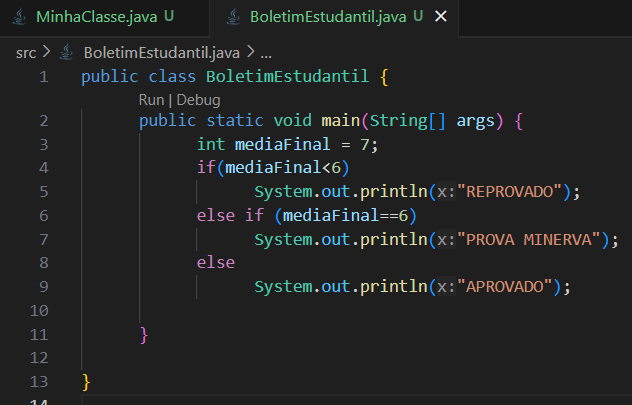
Texto

Descrição gerada automaticamente

BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 4 – Anatomia das classes - 4

Identação

Basicamente indentar é um termo utilizado para escrever o código do programa de forma hierárquica, facilitando assim a visualização e o entendimento do programa.



BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 5 – Anatomia das classes - 5

Organizando arquivos

À medida que nosso sistema vai evoluindo, surgem novos arquivos (código fonte) em nossa estrutura de arquivos do projeto. Isso exige que seja realizado uma organização destes arquivos através de pacotes (packages).

Tela de computador com letras e números em fundo preto

Descrição gerada automaticamente

Os pacotes são criados em forma de diretórios e subdiretórios de forma sequencial.

BLOCO 1 | CURSO 3 | SEÇÃO 2 | AULA 6 – Anatomia das classes - 6

Java Beans

Uma das maiores dificuldades na programação é escrever algoritmos legíveis a níveis que sejam compreendidos por todo seu time ou por você mesmo no futuro. Para isso a linguagem Java sugere, através de convenções, formas de escrita universal para nossas classes, atributos, métodos e pacotes.

BLOCO 1 | CURSO 3 | SEÇÃO 3 | AULA 1 – Tipos de variáveis - 1

Java Sintaxe

Tipos e variáveis

Explorar os tipos de dados.

- Numéricos inteiros e decimais

- Tipos lógicos

- Caracteres

- Objetos

- Diferença entre variável e constante.

BLOCO 1 | CURSO 3 | SEÇÃO 3 | AULA 2 – Tipos de variáveis - 2

Criando mais um repositório no GitHub para atualização da pasta.

Fiz atualização na pasta local e gerenciei com a pasta do GitHub

BLOCO 1 | CURSO 3 | SEÇÃO 3 | AULA 3 – Tipos de variáveis - 3

Tipos e variáveis

Tipos de dados

No Java, existem algumas palavras reservadas para a representação dos tipos de dados básicos que precisam ser manipulados para a construção de programas. Estes tipos de dados são conhecidos como tipos primitivos (Primitive Types).

Os oitos tipos primitivos em Java são:

Int | byte | short | long | float | double | boolean | char

Esses tipos não são considerados objetos, e, portanto, representam valores brutos. Eles são armazenados diretamente na pilha de memória.

Byte | int | short | long são variáveis do tipo INTEIRO.

Declaração de variáveis

Variável é uma identificação de um espaço em memória utilizado pelo nosso programa. Seguindo as convenções em linguagem de programação, toda variável é composta por: tipos de dados + identificação + valor atribuído.

Obs.: Existem algumas peculiaridades a trabalhar com alguns tipos específicos.

Quando usamos float ou long colocamos F e L no final das variáveis respectivamente. Ou seja, o tipo long precisa terminar com L, o tipo float precisa terminar com F e alguns cenários do dia a dia podem estimular uma alteração de tipos de dados convencional.



BLOCO 1 | CURSO 3 | SEÇÃO 3 | AULA 4 – Tipos de variáveis - 4

Constantes

São valores armazenados em memória que não podem ser modificados depois de declarados. Em Java, esses valores são representados pela palavra reservada final, seguida do tipo.

Por convenção, todo valor de uma variável que queiramos que se torne uma constante, o seu nome deve ser escrita em CAIXA ALTA.

Para transformar uma variável em constante temos que seguir a estrutura:

*Final* tipo NOME = valor

BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 1 – Operadores – 1

Tipos de operadores

Objetivo dessa seção: Apresentar o significado dos símbolos especiais de acordo com a linguagem de programação Java.

- Operadores aritméticos

- Operadores unários

- Operadores relacionais

- Operadores lógicos

- Operador ternário

BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 2 – Operadores – 2

Operadores

Símbolos especiais que tem um significado próprio para a linguagem e estão associados a determinadas operações.

Atribuição ( = )

É utilizado para definir o valor inicial ou sobrescrever o valor de uma variável. Em java, definimos um tipo, nome e opcionalmente atribuímos um valor à variável através do operador de atribuição.

Ex.:

String nome = “Leonardo”;

Int idade = 22;

Double peso = 68.5;

Char sexo = ‘M’;

Boolean doadorOrgao = false;

Date dataNascimento = new Date(). (aqui é uma classe que é usada como uma variável e usando a palavra new)

Aritméticos

Utilizado para realizar operações matemáticas entre valores numéricos, podendo se tornar ou não uma expressão mais complexa.

Adição | Subtração | Multiplicação | Divisão

Obs.: O operador de adição (+), quando utilizado em variáveis do tipo texto, realizará a concatenação de textos.

Na imagem abaixo foi feita uma simulação de debug utilizando ponto de parada (breakpoint)

Texto

Descrição gerada automaticamente

BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 3 – Operadores – 3

Unários

Esses operadores são aplicados juntamente com um outro operador aritmético. Eles realizam alguns trabalhos básicos como incrementar, decrementar, inverter valores numéricos e booleanos.

( + ) Operador unário de valor positivo

( - ) Operador unário de valor negativo

(++) Operador unário de incremento de valor

(--) Operador unário de decremento de valor

( ! ) Operador unário de negação

BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 4 – Operadores – 4

Foi feito um exemplo de incrementação e decrementação de valor na variável.

Também a possibilidade de inversão dos valores booleanos com a negação (!)

Texto

Descrição gerada automaticamente

BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 5 – Operadores – 5

Operador ternário

O operador de condição ternária é uma forma resumida para definir uma condição e escolher por um dentre dois valores. Devemos pensar numa condição ternário como se fosse uma condição IF normal, porém, de uma forma em que toda a estrutura estará escrita numa única linha.

Sintaxe:

<expressão condicional> **?** <caso condição seja true> **:**  <caso condição seja false>

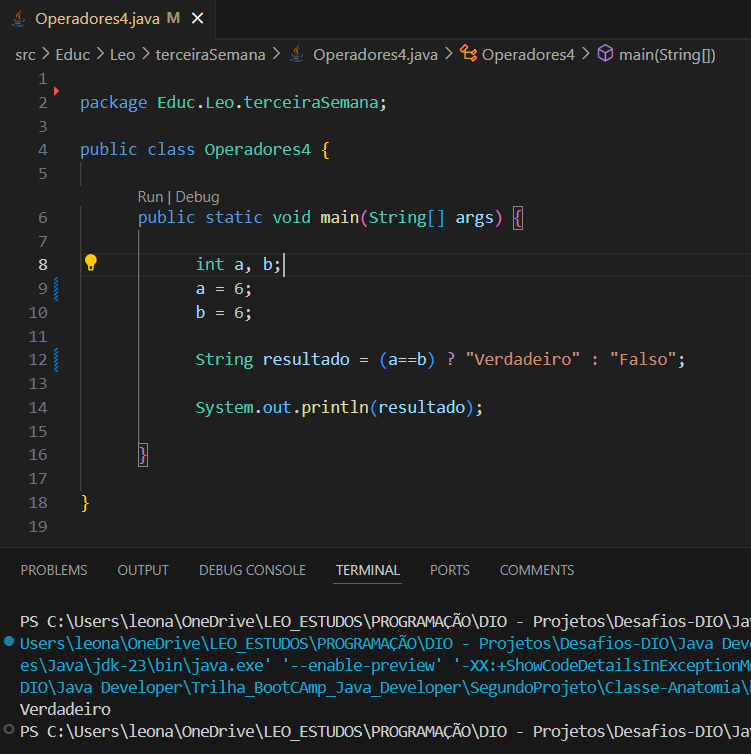
Vou usar o controle de fluxo abaixo para exemplificar

Texto

Descrição gerada automaticamente

Transformei a expressão acima em um operador ternário:

Deixando a aplicação em uma única linha de código.



BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 6 – Operadores – 6

Operadores relacionais

Esses tipos de operadores avaliam a relação entre duas variáveis ou expressões. Neste caso, mais precisamente, definem se o operando à esquerda é igual, diferente, menor, menor ou igual, maior ou igual ao da direta, retornando um valor booleano como resultado.

(==) Verifica se uma variável é igual a outra;

!= Verifica se uma variável é diferente a outra;

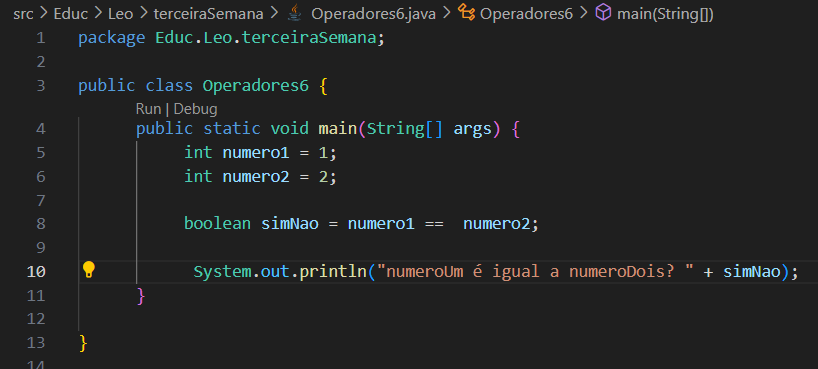
> Verifica se uma variável é maior que a outra;

>= Verifica se uma variável é maior ou igual que a outra;

< Verifica se uma variável é menor que a outra;

<= Verifica se uma variável é menor ou igual que a outra.

Exemplo de aplicação de operador relacional:

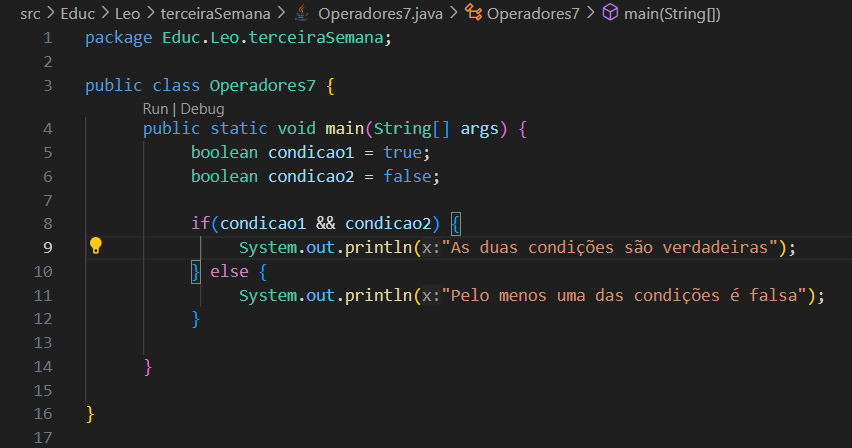


BLOCO 1 | CURSO 3 | SEÇÃO 4 | AULA 7 – Operadores – 7

Operadores lógicos representam o recurso que nos permite criar expressões lógicas maiores a partir da junção de duas ou mais expressões.

&& operador lógico “E”

|| operador lógico “OU”



BLOCO 1 | CURSO 3 | SEÇÃO 5 | AULA 1 – Métodos – 1

Sintaxe com ênfase em métodos

Critérios para criação de métodos, parâmetros e visibilidade.

- Qual a proposta principal de um método¿

- Qual o tipo de retorno esperado após executar o método¿

- Quais os parâmetros serão necessários para a execução do método¿

- O método possui o risco de apresentar alguma execução¿

- Qual a visibilidade do método¿

BLOCO 1 | CURSO 3 | SEÇÃO 5 | AULA 2 – Métodos – 2

Todas as ações das aplicações são consideradas métodos.

Uma classe é definida por atributos e métodos. Já vimos que atributos são, em sua grande maioria, variáveis de diferentes tipos e valores. Os métodos, por sua vez, correspondem a funções ou sub-rotinas disponíveis dentro nossas classes.

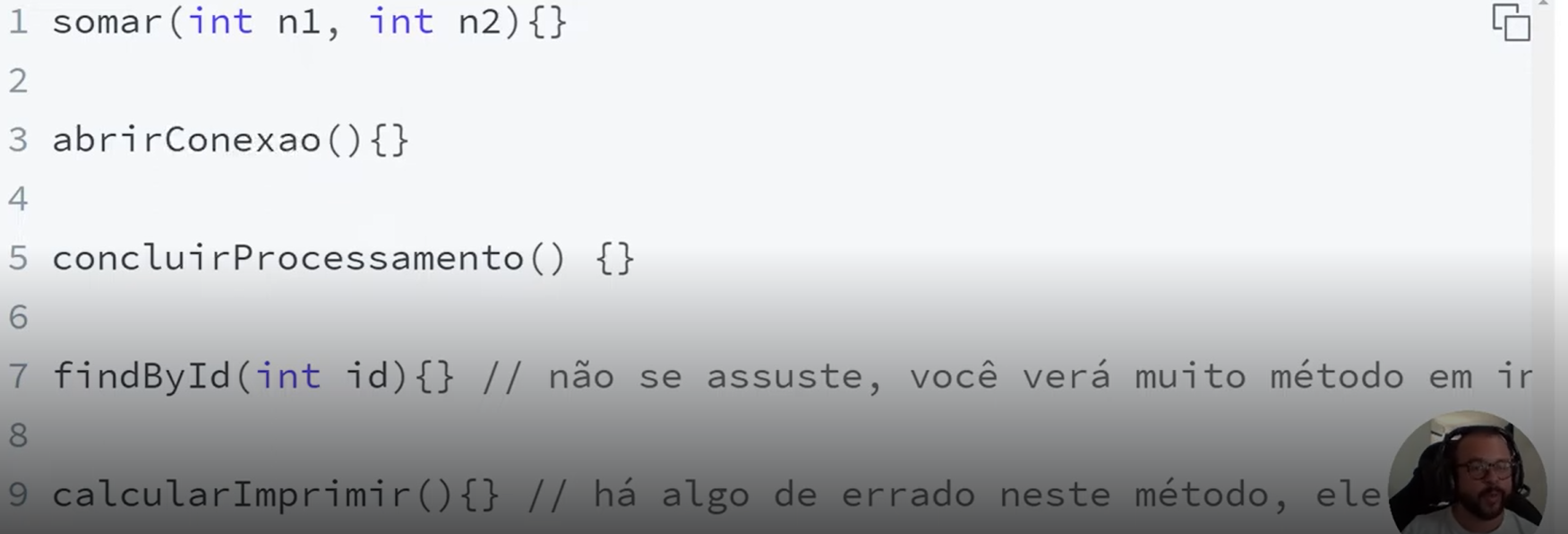
Critério de nomeação de métodos

Esses critérios não são obrigatórios, mas é recomendável que sejam seguidos, pois essas convenções facilitam a vida dos programadores ao trabalharem em códigos de forma colaborativa:

- Deve ser nomeado como verbo;

- Seguir o padrão camelCase (Todas as letras minúsculas com a exceção da primeira da segunda palavra.

Exemplos:



Não existe em Java o conceito de métodos globais. Todos os métodos devem sempre ser definidos dentro de suas classes.

Caso o método não retorne nenhum valor, ele será representado pela palavra-chave void.

Estrutura ex.:

Public double somar (int num1, int num2) {

// lógica aplicada – finalidade do método...

Return...;

}

Public void imprimir(String texto) {

// lógica aplicada – finalidade do método...

// aqui não precisa de return pois

// não será retornado nenhum resultado

}

BLOCO 1 | CURSO 3 | SEÇÃO 5 | AULA 3 – Métodos – 3

Exercitando

Vamos criar um exemplo de uma classe para representar uma SmartTV:

1 – Tenha as características: ligada (boolean), canal(int) e volume(int);

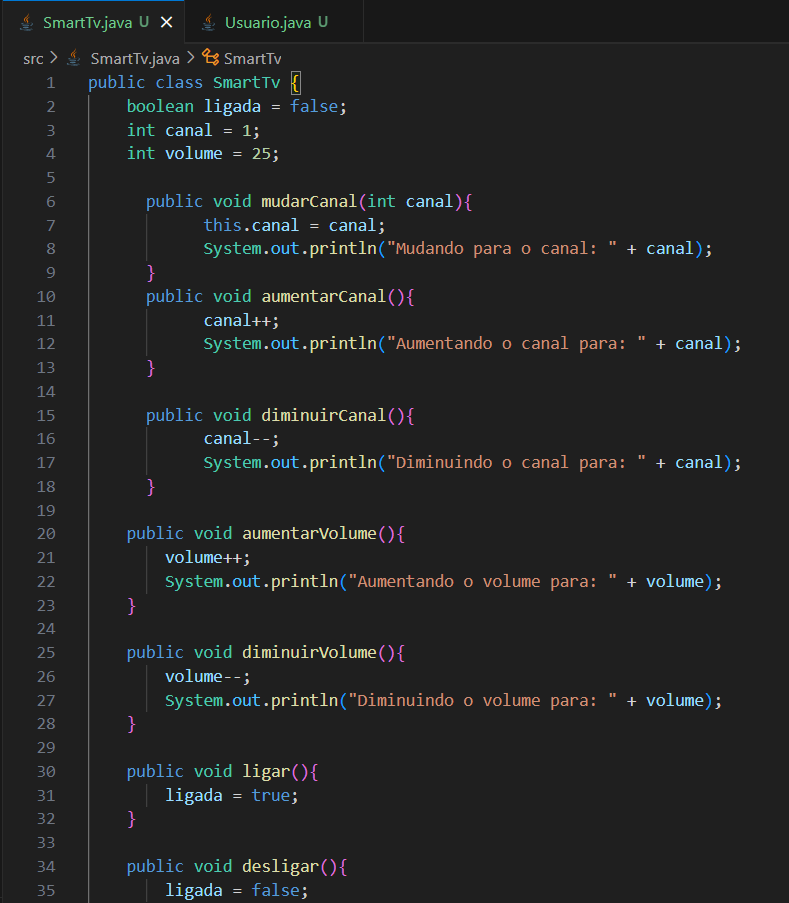
2 – Ligar e desligar;

3 – Aumentar e diminuir o volume sempre em +1 ou -1;

4 – Mudar de canal de 1 em 1 ou definindo o número correspondente;

O projeto está locado no diretório:

C:\Users\leona\OneDrive\LEO\_ESTUDOS\PROGRAMAÇÃO\DIO - Projetos\Desafios-DIO\Java Developer\Trilha\_BootCAmp\_Java\_Developer\TerceiroProjeto

 Texto

O conteúdo gerado por IA pode estar incorreto.

BLOCO 1 | CURSO 3 | SEÇÃO 6 | AULA 1 – Escopo – 1

- Escopos de classe;

- Escopo de método;

- Escopo de fluxo;

BLOCO 1 | CURSO 3 | SEÇÃO 6 | AULA 2 – Escopo – 2

O escopo pode ser entendido como o ambiente onde uma variável pode ser acessada. Em Java, o escopo de variáveis vai de acordo com o bloco onde ela foi declarada.

A variável criada no primeiro acesso a ela, se tornando inacessível após o interpretador sair do bloco de execução ao qual ela pertence. Portanto, esta variável não pode ser lida ou manipulada por rotinas e códigos que estão fora do seu bloco de declaração, ou seja, fora do escopo da variável.

Em uma classe, podemos visualizar a diferença de escopos. Os atributos (variáveis) são declarados no corpo principal da classe, sendo, portanto, acessíveis por todos os métodos.

Caso declare uma variável dentro de um método, o escopo dessa variável está limitado apenas ao corpo desse método, ou seja, dentro, dentro das chaves que limitam os método.

Uma parte fundamental na elaboração de algoritmos simples ou complexos é determinar a localização do código em questão. Sem um domínio sobre escopo de códigos seu projeto tende a conter falhas estruturais e comprometer a proposta principal da aplicação.

BLOCO 1 | CURSO 3 | SEÇÃO 7 | AULA 1 – Palavras Reservadas – 1

É preciso explorar a real finalidade do uso das palavras reservadas.

52 Palavras reservadas e organizadas por classificação de usabilidade considerando as regras da linguagem.

Assuntos;

- Tipos primitivos;

- Classificações;

- Escopo de uso;

- Palavras “opostas”.

BLOCO 1 | CURSO 3 | SEÇÃO 7 | AULA 2 – Palavras Reservadas – 2

São identificadores de uma linguagem que já possuem uma finalidade específica, portanto não podem ser utilizados para nomear variáveis, classes, métodos ou atributos.

A linguagem Java possui 53 palavras reservadas. Todas essas palavras são classificadas em grupos e escritas com letra minúscula, sendo identificadas com uma cor especial pela maioria das IDE’s.

Palavras Reservadas:

Controle de pacotes

**Import**: importa pacotes ou classes para dentro do código;

**Package**: especifica a que pacote todas as classes de um arquivo pertencem;

Modificadores de acesso

**Public**: acesso de qualquer classe;

**Private**: acesso apenas na classe;

**Protected**: acesso por classes no mesmo pacote e subclasses;

Primitivos

**Boolean**: um valor indicando verdadeiro ou falso;

**Byte**: um inteiro de 8 bits;

**Char**: um character Unicode (16-bit unsigned);

**Double**: um número de ponto flutuante de 64 bits (signed);

**Float**: um número de ponto flutuante de 32 bits (signed);

**Int**: um inteiro de 32 bits;

**Long**: um inteiro de 64 bits;

**Short**: um inteiro de 32 bits;

Modificadores de classes, variáveis ou métodos

**Abstract**: classe que não pode ser instanciada ou método que precisa ser implantado por uma subclasse não abstrata;

**Class**: especifica uma classe;

**Extends**: indica a superclasse que a subclasse está estendendo;

**Final**: impossibilita que uma classe seja estendida, que um método seja sobrescrito ou que uma variável seja reinicializada;

**implements:** indica as interfaces que uma classe irá implementar

**interface:** especifica uma interface

**native:** indica que um método está escrito em uma linguagem dependente de plataforma, como o C

**new:** instancia um novo objeto, chamando seu construtor

**static:** faz um método ou variável pertencer à classe ao invés de instanciá-las

**strictfp:** usado em frente a um método ou classe para indicar que os números de ponto flutuante seguirão as regras de ponto flutuante em todas as expressões

**synchronized:** indica que um método só pode ser acessado por uma thread de cada vez

**transient:** impede a serialização de campos

**volatile:** indica que uma variável pode ser alterada durante o uso de threads

**Controle de fluxo dentro de um bloco de código**

**break:** sai do bloco de codigo em que ele está

**case:** executa um bloco de código dependendo do teste do switch

**continue:**pula a execução do código que viria após essa linha e vai para a próxima passagem do loop

**default:**executa esse bloco de codigo caso nenhum dos teste de switch-case seja verdadeiro

**do:** executa um bloco de código uma vez, e então realiza um teste em conjunto com o while para determinar se o bloco deverá ser executado novamente

**else:** executa um bloco de código alternativo caso o teste if seja falso

**for:** usado para realizar um loop condicional de um bloco de código

**if:** usado para realizar um teste lógico de verdadeiro o falso

**instanceof:** determina se um objeto é uma instância de determinada classe, superclasse ou interface

**return:** retorna de um método sem executar qualquer código que venha depois desta linha (também pode retornar uma variável)

**switch:**indica a variável a ser comparada nas expressões case

**while:** executa um bloco de código repetidamente enquanto a condição for verdadeira

**Tratamento de erros**

**assert:** testa uma expressão condicional para verificar uma suposição do programador

**catch:** declara o bloco de código usado para tratar uma exceção

**finally:**bloco de código, após um try-catch, que é executado independentemente do fluxo de programa seguido ao lidar com uma exceção

**throw:**usado para passar uma exceção para o método que o chamou

**throws:** indica que um método pode passar uma exceção para o método que o chamou

**try:** bloco de código que tentará ser executado, mas que pode causar uma exceção

**Variáveis de referência**

**super:** refere-se a superclasse imediata

**this:** refere-se a instância atual do objeto

**Retorno de um método**

**void:** indica que o método não tem retorno

**Palavras reservadas não utilizadas**

**const:** Não utilize para declarar constantes; use public static final

**goto:** não implementada na linguagem Java por ser considerada prejudicial

**Literais reservados**

De acordo com a Java Language Specification, null, true e false são tecnicamente chamados de valores literais, e não keywords. Se você tentar criar algum identificador com estes valores, você também terá um erro de compilação.

BLOCO 1 | CURSO 3 | SEÇÃO 8 | AULA 1 – Java DOC – 1

Objetivo: Apresentar a ferramenta de documentação da Linguagem Java e explorar como adicionar comentários em nossos arquivos.

Assuntos:

- Documentação oficial;

- Tags;

- Tipos de comentários;

- Javadoc.

BLOCO 1 | CURSO 3 | SEÇÃO 8 | AULA 2 – Java DOC – 2

Uma das maiores características da linguagem Java é que desde suas primeiras versões tínhamos um nossas mãos uma documentação rica e detalhada dos recursos da linguagem.

Conforme site oficial, podemos compreender e explorar todos os recursos organizados por pacotes e classes bem específicas sem nem mesmo escrever uma linha de código.

Hoje costuma-se afirmar que para se tornar um desenvolvedor nível avançado, é requisito imprescindível adquirir a habilidade de compreender a documentação oficial da linguagem e dos frameworks que são incorporados nos projetos atuais.