

PROGRAMANDO O AMANHÃ COM JAVA

```
public class HelloWorld {  
    {  
        public static void main(String  
            [] args) {  
            System.out.println('Hello,  
                Java!');  
        }  
    }  
}
```

IGOR LUNA

Sumário

1. Introdução ao Java: História – pág. 3
2. Preparando Seu Ambiente de Programação – pág. 5
3. Hello World: Primeiro programa – pág. 7
4. Manipulando variáveis e constantes – pág. 9
5. Import de bibliotecas – pág. 12
6. Manipulando entradas de dados – pág. 14
7. Tipos de dados em Java – pág. 18
8. Tipos de operadores – pág. 22
9. Condicionais: If, Else if, Else if – pág. 28
10. Estruturas de repetição – pág. 31
11. Conclusão: Agradecimentos – pág. 35



01

Introdução ao Java: História

Introdução ao Java

História

Java é uma linguagem de programação criada por James Gosling e a equipe da Sun Microsystems em 1995. Projetada para ser simples, segura, orientada a objetos e portátil, ela permite que programas sejam executados em diferentes sistemas operacionais usando a Java Virtual Machine (JVM). O lema “escreva uma vez, execute em qualquer lugar” reflete bem essa filosofia.

Popular em aplicações corporativas, desenvolvimento de aplicativos Android, sistemas embarcados e soluções web, Java é amplamente usada por empresas como Amazon, LinkedIn e Oracle. Sua robustez e grande ecossistema a tornaram uma das linguagens mais empregadas no mundo da tecnologia.

Se você quer construir aplicativos multiplataforma, desenvolver para Android ou trabalhar em sistemas de grande porte, Java é uma escolha sólida para quem deseja se aprofundar no universo da programação.

02

Preparando Seu Ambiente de Programação

Preparando seu ambiente de programação

Ambiente

Antes de começar a programar em Java, é essencial configurar corretamente o ambiente de desenvolvimento. Isso garante que você consiga escrever, compilar e executar seus códigos de forma eficiente.

O primeiro passo é instalar o Java Development Kit (JDK), que contém as ferramentas necessárias para compilar e rodar programas Java. Você pode baixá-lo no site oficial da Oracle ou utilizar o OpenJDK. Após a instalação, verifique se o Java está funcionando corretamente digitando `java -version` no terminal ou prompt de comando.

Em seguida, escolha um editor ou IDE (Ambiente de Desenvolvimento Integrado). As opções mais comuns são Visual Studio Code, IntelliJ IDEA, Eclipse e NetBeans. Elas oferecem recursos que facilitam a escrita de código, como destaque de sintaxe, sugestões automáticas e execução simplificada.

Com o JDK e a IDE instalados, você já pode criar e executar seu primeiro programa. Manter uma boa organização de pastas e arquivos desde o início ajuda a manter seu projeto limpo e bem estruturado.

03

**Hello World:
Primeiro
programa**

Hello World

Primeiro programa

A estrutura básica começa com a definição da classe (public class HelloWorld), seguida pelo método principal (public static void main(String[] args)), que é o ponto de entrada do programa.

Dentro dele, usamos System.out.println("Hello World!"); para exibir uma mensagem na tela. É o famoso "Olá, Mundo!", tradicional entre iniciantes.



```
1  /*Classe */
2  public class HelloWorld {
3      /* Método principal da classe */
4      public static void main(String[] args){
5
6          System.out.println("Hello World!"); //Imprime a mensagem "Hello World!"
7
8      } //fim do método
9
10 } //fim da classe
```


04

Manipulando variáveis e constantes

Manipulando variáveis e constantes

Variáveis

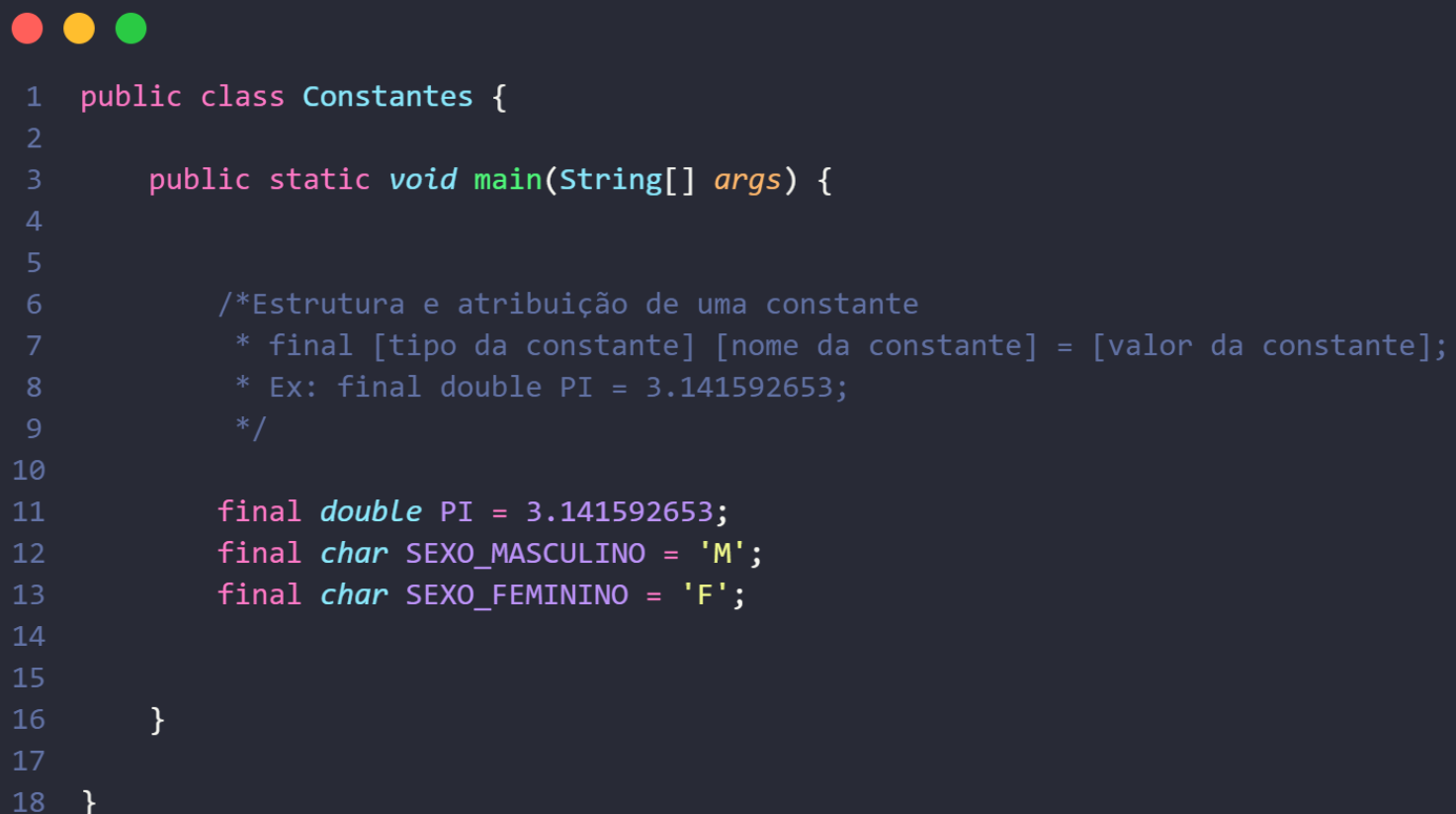
Variáveis são espaços reservados na memória do computador usados para armazenar informações que podem mudar durante a execução do programa. Elas permitem guardar e manipular dados como números, textos ou valores lógicos ao longo do código.

```
1  public class Variavel {
2
3      public static void main(String[] args) {
4
5          /*Estrutura de uma variável
6           * [tipo da variavel] [nome da variavel];
7           * Ex: int idade;
8           *
9           * Atribuindo um valor
10          * [tipo da variavel] [nome da variavel] = [valor];
11          * Ex: int idade = 19;
12          */
13
14          String nome = "Igor";
15          int idade = 31;
16          boolean casado = false;
17
18          //Variável de referência
19          idade = 19; //A variável idade tinha o valor 31, agora foi atribuido o valor 19
20          casado = true; //A variável casado tinha o valor false, agora foi atribuido o valor true
21
22          System.out.println(nome); //imprime o valor alocado na variável nome = "Igor"
23
24      }
25
26  }
```

Manipulando variáveis e constantes

Constantes

Constantes são valores que não mudam durante a execução do programa. Em Java, usamos a palavra-chave `final` para declarar uma constante. Ela é usada quando queremos garantir que uma variável tenha sempre o mesmo valor.



```
1  public class Constantes {
2
3      public static void main(String[] args) {
4
5
6          /*Estrutura e atribuição de uma constante
7           * final [tipo da constante] [nome da constante] = [valor da constante];
8           * Ex: final double PI = 3.141592653;
9           */
10
11         final double PI = 3.141592653;
12         final char SEXO_MASCULINO = 'M';
13         final char SEXO_FEMININO = 'F';
14
15     }
16
17
18 }
```

05

Import de bibliotecas

Import de bibliotecas

Bibliotecas

Bibliotecas em Java são conjuntos de classes prontas que fornecem funcionalidades extras ao programa. Elas ajudam a economizar tempo, evitando que você escreva tudo do zero.

Para usar uma biblioteca, é necessário importá-la no início do código com a palavra-chave `import`.



```
1  /*Estrutura de imports de biblioteca
2   * import [nome do pacote].Classe // caso queira pegar uma classe especifica
3   * ou
4   * import [nome do pacote].* // caso queira pegar todas as classes
5   *
6   * Ex: import java.util.ArrayList;
7   * ou
8   * Ex: import java.util.*;
9   */
10 import java.util.*;
11
12 public class bibliotecas {
13
14     public static void main(String[] args) {
15
16
17     }
18
19 }
```

06

Manipulando entradas de datos

Manipulando entradas de dados

Entradas de dados (Scanner)

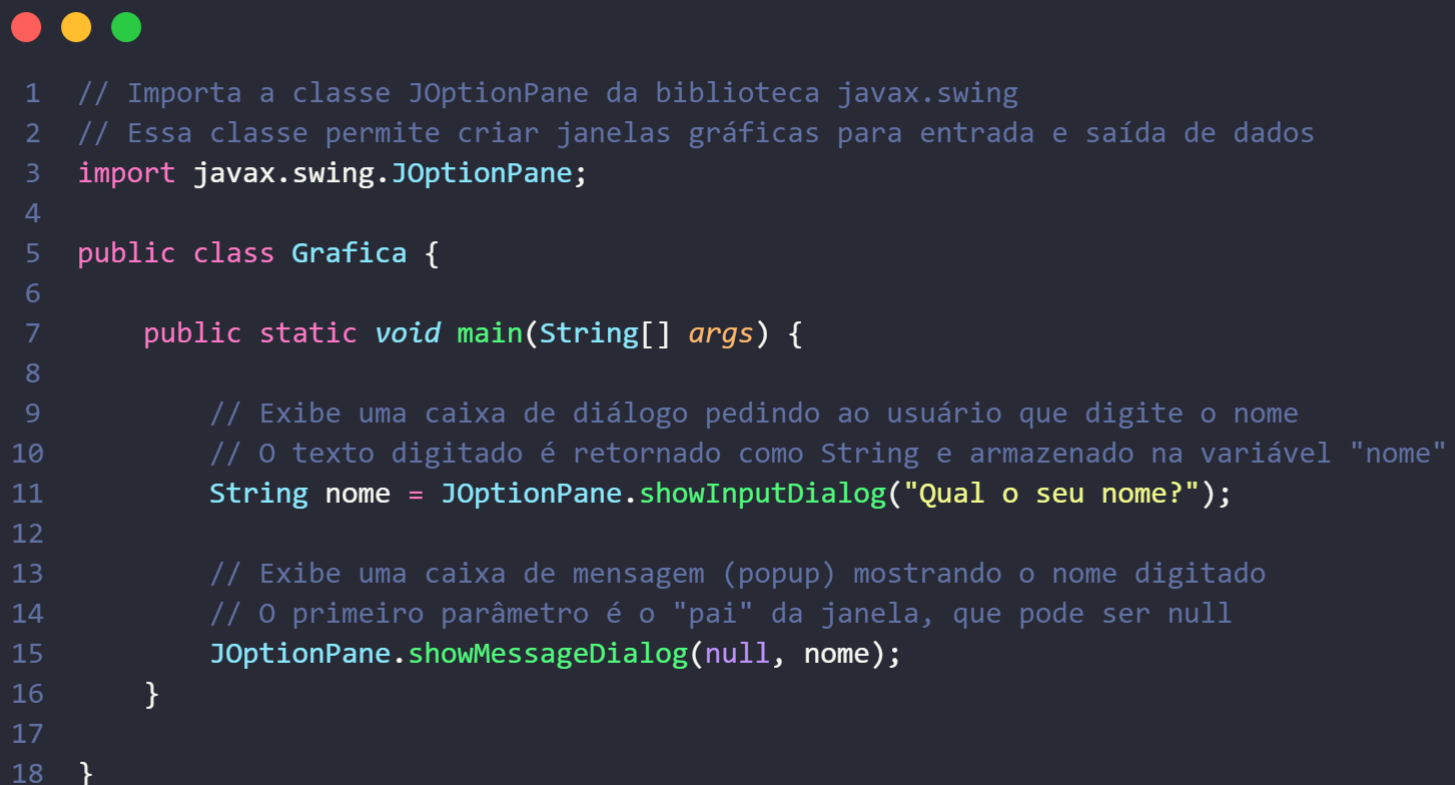
Entrada de dados em Java permite que o usuário digite informações enquanto o programa está sendo executado. Para isso, usamos a classe Scanner, que faz parte da biblioteca java.util.

```
1 // Importa a classe Scanner da biblioteca java.util
2 import java.util.Scanner;
3
4 public class entrada {
5
6     public static void main(String[] args) {
7
8         // Cria um objeto Scanner chamado "s"
9         // new Scanner(System.in) significa: "vou capturar dados digitados no teclado"
10        // System.in é o fluxo de entrada padrão (teclado)
11        Scanner s = new Scanner(System.in);
12        System.out.println("Qual o seu nome?");
13
14        // Lê uma linha digitada pelo usuário e armazena na variável "nome"
15        // nextLine() captura tudo até o Enter ser pressionado
16        String nome = s.nextLine();
17
18        // Exibe uma mensagem de boas-vindas usando o nome digitado
19        System.out.println("Bem vindo " + nome);
20
21    }
22
23 }
```

Manipulando entradas de dados

Entradas e saída de dados (Gráfica)

Entrada e saída de dados gráficas com JOptionPane permitem interações por janelas pop-up, deixando o programa mais visual e amigável. Essa funcionalidade faz parte da biblioteca javax.swing.

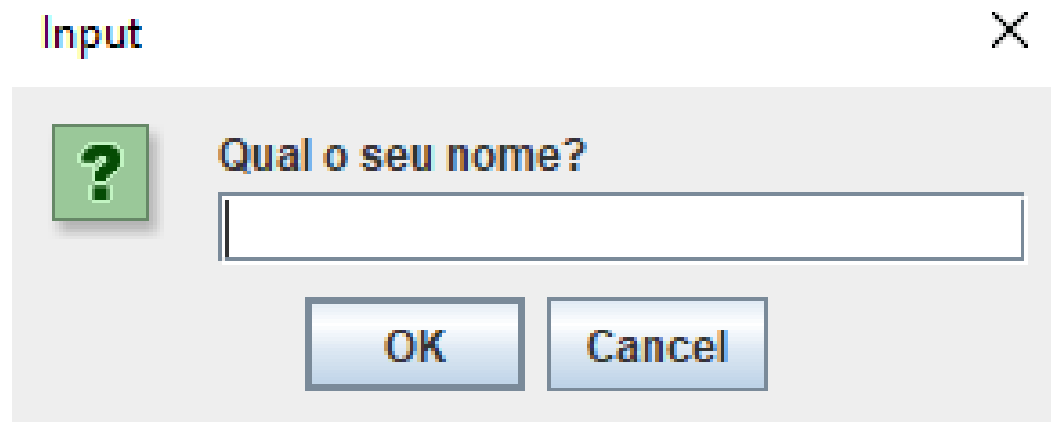


```
1 // Importa a classe JOptionPane da biblioteca javax.swing
2 // Essa classe permite criar janelas gráficas para entrada e saída de dados
3 import javax.swing.JOptionPane;
4
5 public class Grafica {
6
7     public static void main(String[] args) {
8
9         // Exibe uma caixa de diálogo pedindo ao usuário que digite o nome
10        // O texto digitado é retornado como String e armazenado na variável "nome"
11        String nome = JOptionPane.showInputDialog("Qual o seu nome?");
12
13        // Exibe uma caixa de mensagem (popup) mostrando o nome digitado
14        // O primeiro parâmetro é o "pai" da janela, que pode ser null
15        JOptionPane.showMessageDialog(null, nome);
16    }
17
18 }
```

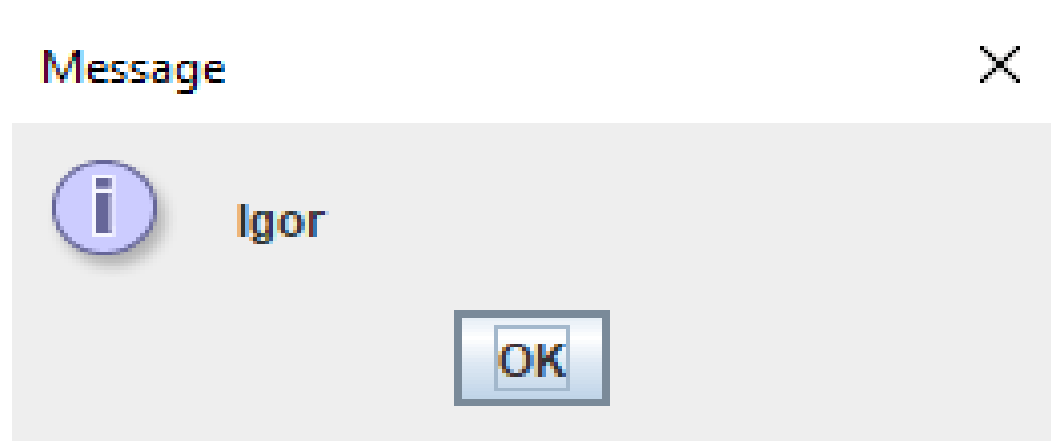
Manipulando entradas de dados

Entradas e saída de dados (Gráfica)

Entrada gráfica:



Saída gráfica:



07

Tipos de dados em java

Tipos de dados em Java

Tipos primitivos

Os tipos primitivos são os mais básicos da linguagem e servem para armazenar dados simples, como números, caracteres e valores lógicos. Cada tipo ocupa uma quantidade específica de memória.

Os tipos primitivos mais usados são `int`, `double`, `char`, `boolean`.



```
1  int idade = 31; //int -> armazena números inteiros
2
3  double preco = 12.45; //double -> armazena números com ponto flutuante, com alta precisão
4
5  char sexo = 'M'; //char -> armazena um único caractere entre aspas simples
6
7  boolean casado = true; //boolean -> armazena verdadeiro ou falso
```

Tipos de dados em Java

Capacidade e forma binária

Byte e short, long são usados quando é necessário controlar melhor o uso da memória.

```
1 byte b = 100; // -128 a 127 (1 byte)
2 short s = 32000; // -32.768 a 32.767 (2 bytes)
3 int i = 2000000000; // ±2 bilhões (4 bytes)
4 long l = 9000000000000000000L; //±9 quintilhões (8 bytes)
5 //L no final do long indica que é um número do tipo long
6
7 //Tipo decimal
8 double d = 1.7976931348623157E+308D; // Valor máximo de double (8 bytes)
9 float f = 123F; // (8 bytes), o F no final indica que é float.
```

Forma binária:

```
1 byte bb = 0b01010101; // 8 bits / 1 byte
2 short ss = 0b0101010101010101; // 16 bits / 2 byte
3 int ii = 0b01010101010101010101010101010101; // 32 bits / 4 byte
4 //0 0b indica que o número está em binário.
```


Tipos de dados em Java

Wrapper

Wrappers em Java são classes que transformam tipos primitivos em objetos, permitindo seu uso em contextos orientados a objetos. Eles também fornecem métodos utilitários para operações como conversão, comparação e análise de valores.

```
1  public class Wrapper {
2
3      public static void main(String[] args) {
4
5          /*Estrutura de um Wrapper
6           * [tipo da classe] [nome da variável];
7           *
8           * Atribuindo valores
9           * [tipo da classe] [nome da variável] = [tipo da classe].valueOf(valor);
10          */
11
12          // Wrappers para tipos numéricos
13          Integer idade = Integer.valueOf(25); // ou: Integer idade = 25;
14          Byte b = Byte.valueOf((byte) 10);
15          Short s = Short.valueOf((short) 32000);
16          Long l = Long.valueOf(9000000000000000000L);
17          Float f = Float.valueOf(12.34F);
18          Double d = Double.valueOf(123.456);
19
20          // Wrapper para char
21          Character sexo = Character.valueOf('M');
22
23          // Wrapper para boolean
24          Boolean ativo = Boolean.valueOf(true);
25
26          // A classe Void é especial: usada apenas para refletir métodos void
27          Void nada = null; // Não instanciamos Void normalmente
28
29      }
30
31  }
32 }
```

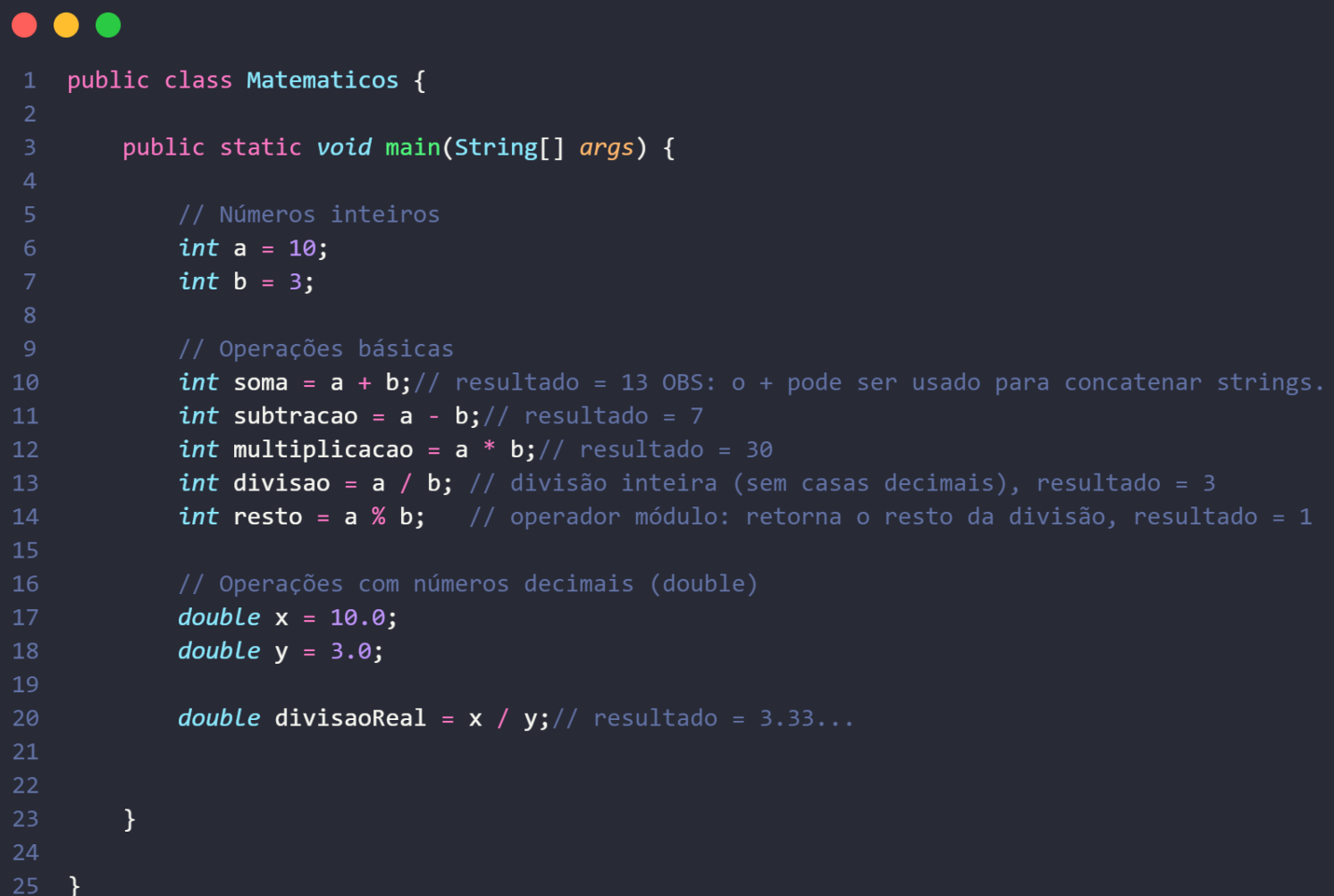
08

Tipos de operadores

Tipos de operadores

Aritméticos

Os operadores aritméticos permitem realizar operações matemáticas básicas entre variáveis ou valores numéricos.

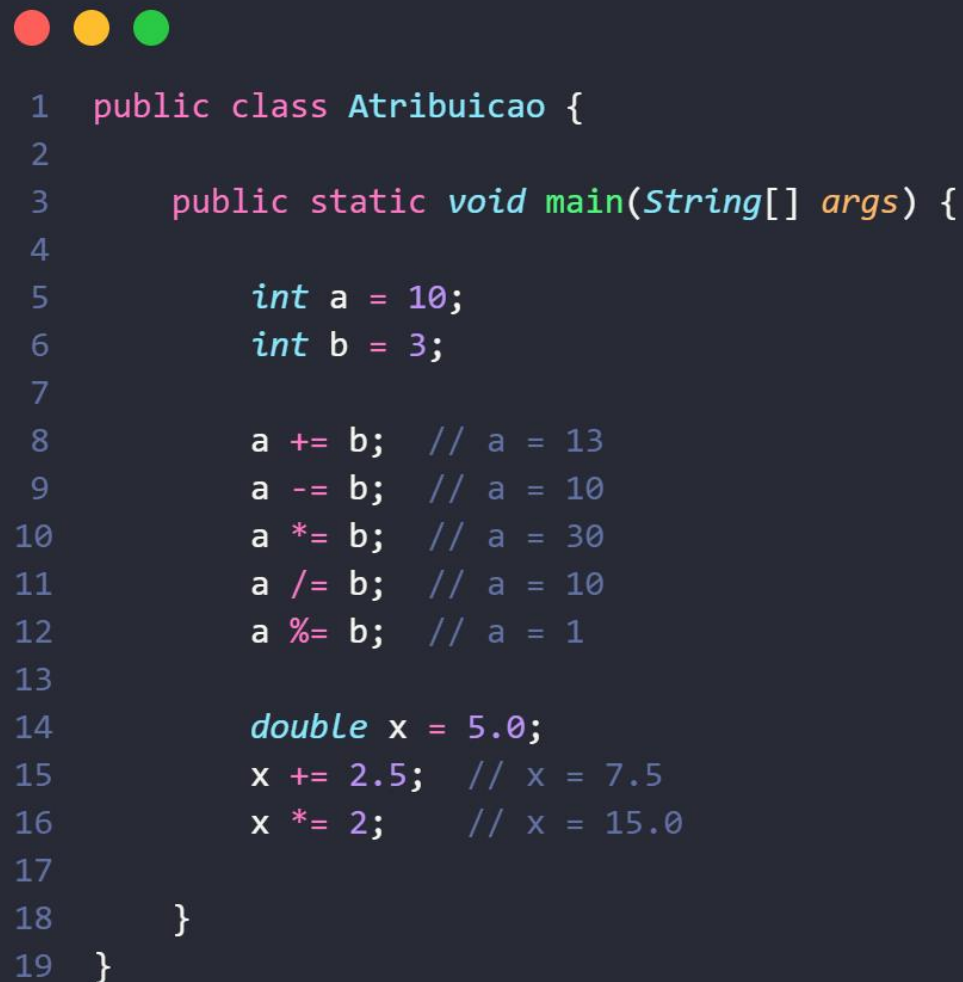


```
1 public class Matematicos {
2
3     public static void main(String[] args) {
4
5         // Números inteiros
6         int a = 10;
7         int b = 3;
8
9         // Operações básicas
10        int soma = a + b; // resultado = 13 OBS: o + pode ser usado para concatenar strings.
11        int subtracao = a - b; // resultado = 7
12        int multiplicacao = a * b; // resultado = 30
13        int divisao = a / b; // divisão inteira (sem casas decimais), resultado = 3
14        int resto = a % b; // operador módulo: retorna o resto da divisão, resultado = 1
15
16        // Operações com números decimais (double)
17        double x = 10.0;
18        double y = 3.0;
19
20        double divisaoReal = x / y; // resultado = 3.33...
21
22
23    }
24
25 }
```

Tipos de operadores

Atribuição

Os operadores de atribuição são usados para atribuir valores a variáveis, com possibilidade de fazer uma operação ao mesmo tempo.

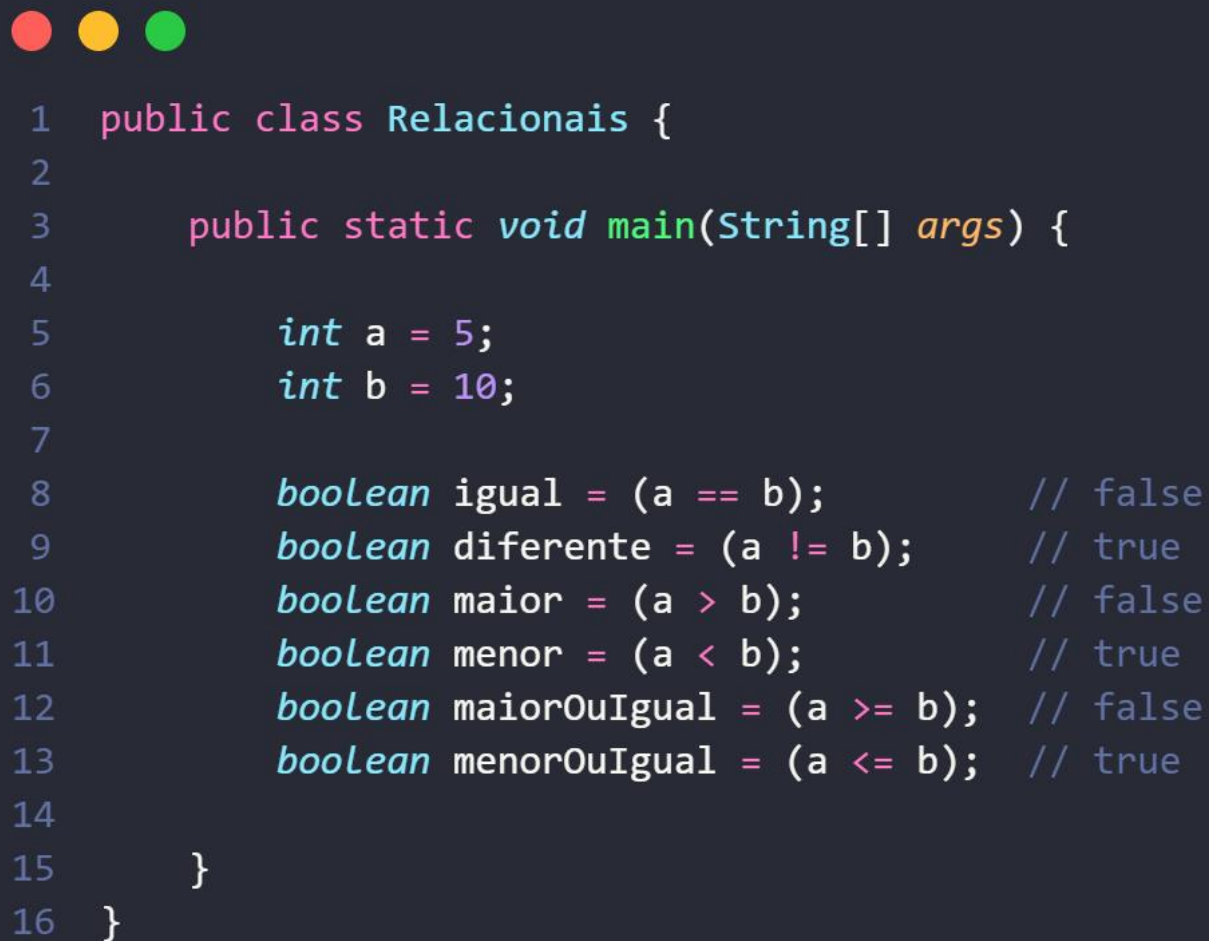


```
1  public class Atribuicao {  
2  
3      public static void main(String[] args) {  
4  
5          int a = 10;  
6          int b = 3;  
7  
8          a += b; // a = 13  
9          a -= b; // a = 10  
10         a *= b; // a = 30  
11         a /= b; // a = 10  
12         a %= b; // a = 1  
13  
14         double x = 5.0;  
15         x += 2.5; // x = 7.5  
16         x *= 2; // x = 15.0  
17  
18     }  
19 }
```

Tipos de operadores

Relacionais

Os operadores relacionais são usados para comparar dois valores. O resultado é sempre true ou false.

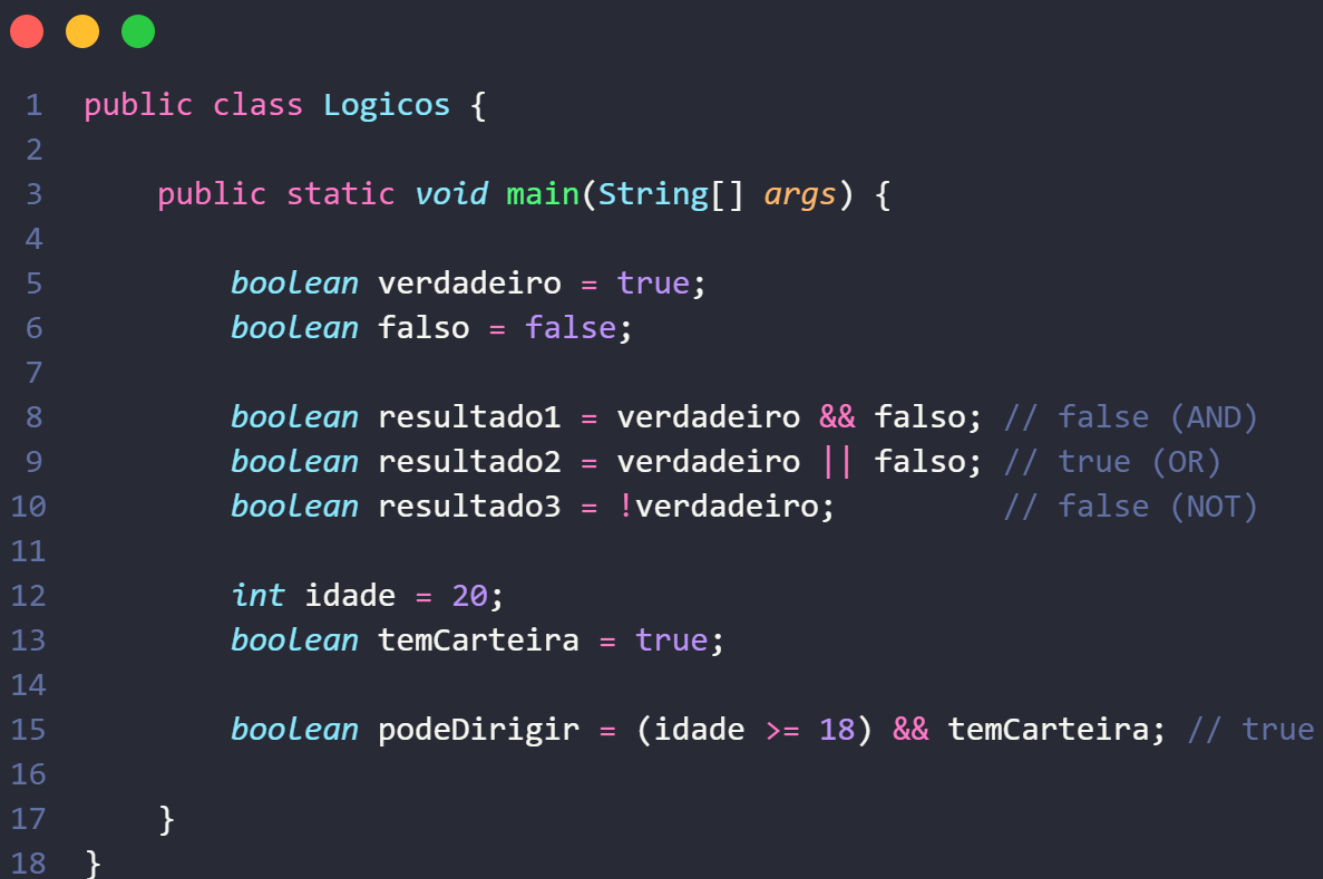


```
1  public class Relacionais {  
2  
3      public static void main(String[] args) {  
4  
5          int a = 5;  
6          int b = 10;  
7  
8          boolean igual = (a == b);           // false  
9          boolean diferente = (a != b);       // true  
10         boolean maior = (a > b);             // false  
11         boolean menor = (a < b);             // true  
12         boolean maiorOuIgual = (a >= b);     // false  
13         boolean menorOuIgual = (a <= b);     // true  
14  
15     }  
16 }
```

Tipos de operadores

Lógicos

Os operadores lógicos são utilizados para combinar expressões booleanas, ou seja, que retornam true ou false.



```
1 public class Logicos {  
2  
3     public static void main(String[] args) {  
4  
5         boolean verdadeiro = true;  
6         boolean falso = false;  
7  
8         boolean resultado1 = verdadeiro && falso; // false (AND)  
9         boolean resultado2 = verdadeiro || falso; // true (OR)  
10        boolean resultado3 = !verdadeiro;          // false (NOT)  
11  
12        int idade = 20;  
13        boolean temCarteira = true;  
14  
15        boolean podeDirigir = (idade >= 18) && temCarteira; // true  
16  
17    }  
18 }
```


Tipos de operadores

Ternário

Os operadores ternários servem para é um atalho para escrever um if-else simples em uma única linha.



```
1 // Define a variável idade com valor 18
2 int idade = 18;
3
4 // Usa o operador ternário para verificar se idade é maior ou igual a 18
5 // Se for, status recebe "Maior de idade", senão recebe "Menor de idade"
6 String status = (idade >= 18) ? "Maior de idade" : "Menor de idade";
7 // Saída: status = "Maior de idade"
```



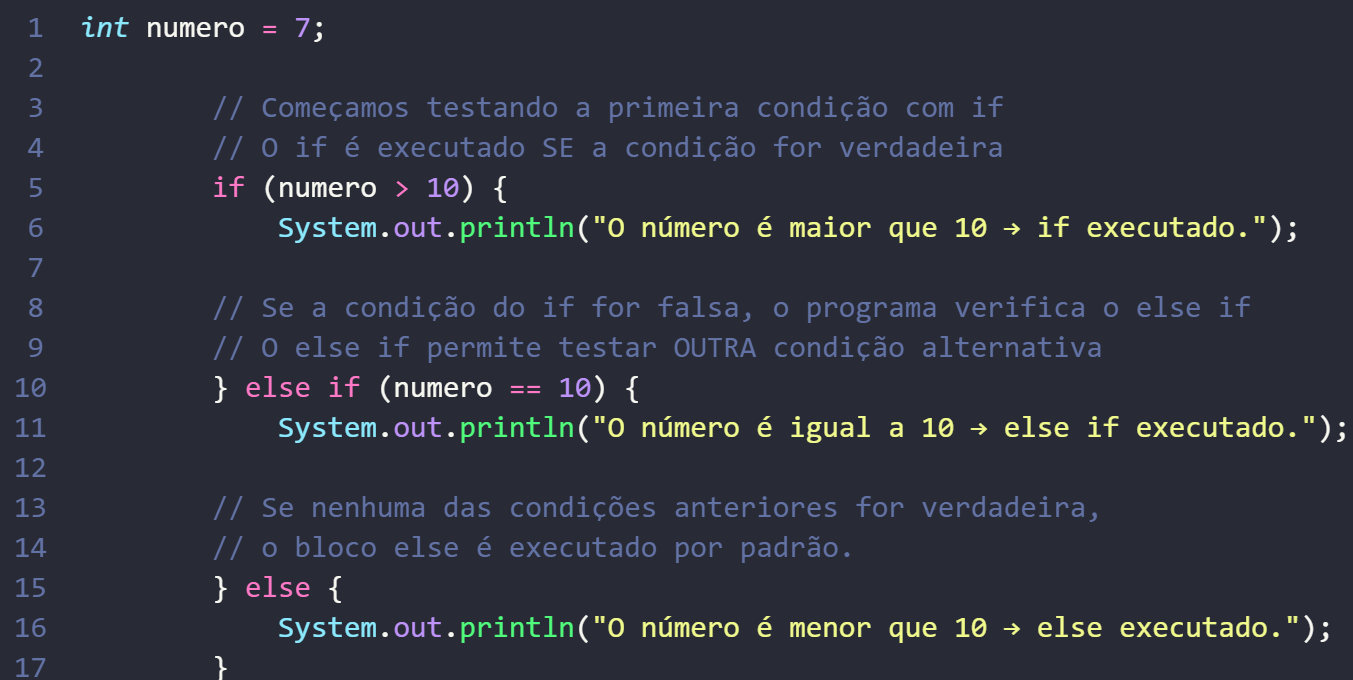
09

**Condicionais: if,
else, else if**

Condicionais: if, else, else if

If, else, else if

Em Java, if, else if e else são usados para tomar decisões no programa. O if verifica uma condição; se for verdadeira, executa um bloco de código. O else if permite testar outras condições caso o if falhe. Já o else executa um bloco final se nenhuma das anteriores for verdadeira.



```
1  int numero = 7;
2
3  // Começamos testando a primeira condição com if
4  // O if é executado SE a condição for verdadeira
5  if (numero > 10) {
6      System.out.println("O número é maior que 10 → if executado.");
7
8  // Se a condição do if for falsa, o programa verifica o else if
9  // O else if permite testar OUTRA condição alternativa
10 } else if (numero == 10) {
11     System.out.println("O número é igual a 10 → else if executado.");
12
13 // Se nenhuma das condições anteriores for verdadeira,
14 // o bloco else é executado por padrão.
15 } else {
16     System.out.println("O número é menor que 10 → else executado.");
17 }
```

Condicionais: if, else, else if

Exemplos:

Outro com inteiros:

```
1 // Estrutura com múltiplos else if
2 if (idade < 12) {
3     System.out.println("Criança → entrou no primeiro if.");
4 } else if (idade >= 12 && idade < 18) {
5     System.out.println("Adolescente → entrou no else if.");
6 } else if (idade >= 18 && idade < 60) {
7     System.out.println("Adulto → entrou no segundo else if.");
8 } else {
9     System.out.println("Idoso → nenhum if/else if foi verdadeiro, entrou no else.");
10 }
```

Com booleano:

```
1 // Um exemplo com valor booleano:
2     boolean temChave = false;
3
4     // if sempre testa uma condição verdadeira
5     if (temChave) {
6         System.out.println("Você pode entrar na casa → if executado porque temChave é true.");
7     } else {
8         System.out.println("Você NÃO pode entrar na casa → else executado porque temChave é false.");
9     }
10 }
```

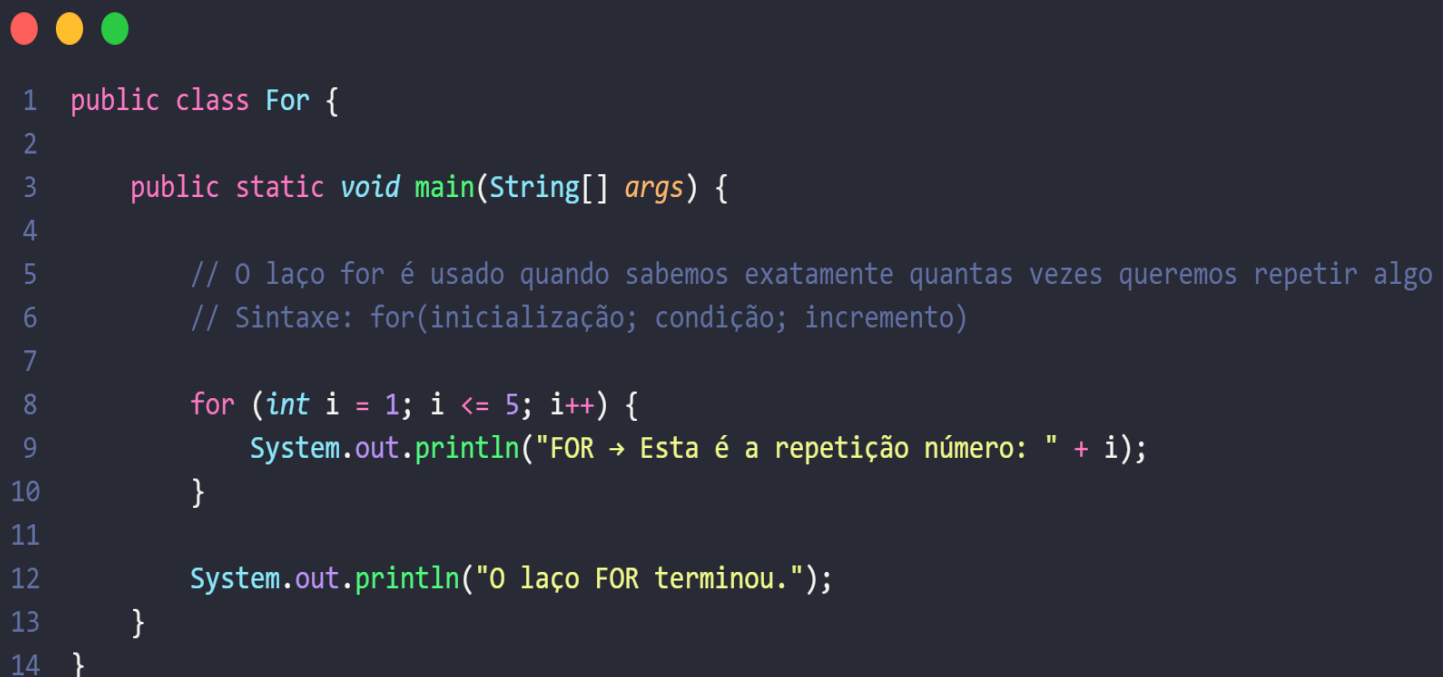
10

Estrutura de Repetição

Estrutura de repetição

For

O laço for é usado quando sabemos exatamente quantas vezes queremos repetir um bloco de código. Ele possui uma estrutura fixa, com três partes: inicialização, condição e incremento. A repetição continua enquanto a condição for verdadeira. É ideal para percorrer intervalos numéricos ou listas.




```
1 public class For {  
2  
3     public static void main(String[] args) {  
4  
5         // O laço for é usado quando sabemos exatamente quantas vezes queremos repetir algo  
6         // Sintaxe: for(inicialização; condição; incremento)  
7  
8         for (int i = 1; i <= 5; i++) {  
9             System.out.println("FOR → Esta é a repetição número: " + i);  
10        }  
11  
12        System.out.println("O laço FOR terminou.");  
13    }  
14 }
```


Estrutura de repetição

While

O while executa um bloco de código enquanto uma condição for verdadeira. Ele verifica a condição antes de entrar no laço, o que significa que, se a condição já for falsa no início, o bloco pode nem ser executado. É indicado quando não sabemos quantas repetições serão necessárias com antecedência.



```
1 public class While {  
2  
3     public static void main(String[] args) {  
4  
5         // O laço while executa ENQUANTO a condição for verdadeira  
6         int contador = 1;  
7  
8         while (contador <= 5) {  
9             System.out.println("WHILE → Repetição número: " + contador);  
10            contador++; // incrementa para não gerar loop infinito  
11        }  
12  
13        System.out.println("O laço WHILE terminou.");  
14    }  
15  
16 }
```

Estrutura de repetição

Do While

O do while é semelhante ao while, mas com uma diferença importante: ele executa o bloco pelo menos uma vez, pois a verificação da condição acontece após a execução. É útil quando queremos garantir que o código dentro do laço seja rodado pelo menos uma vez, mesmo que a condição seja falsa logo depois.

```
1 public class Dowhile {
2
3     public static void main(String[] args) {
4
5         // O do while executa o bloco primeiro, e só DEPOIS verifica a condição
6         int contador = 1;
7
8         do {
9             System.out.println("DO WHILE → Repetição número: " + contador);
10            contador++;
11        } while (contador <= 5);
12
13        System.out.println("O laço DO WHILE terminou.");
14    }
15
16 }
```

11

**Conclusão:
Agradecimentos**

Conclusão

Agradecimentos



Quero agradecer a você, leitor ou leitora, por dedicar seu tempo e interesse para aprender Java com este material. Espero que este eBook tenha sido útil para tornar seus primeiros passos na programação mais leves e claros.

A programação pode parecer desafiadora no começo, mas lembre-se: cada linha de código escrita é um aprendizado, e cada erro, uma oportunidade de melhorar. Obrigado por confiar neste conteúdo para fazer parte da sua jornada!

Desejo muito sucesso nos seus estudos e projetos. Continue explorando, praticando e acreditando no seu potencial!

- Igor Luna

GitHuB: <https://github.com/igorluna06>

Linkedin: <https://www.linkedin.com/in/igor-luna-87b813303/>

