M01

Estrutura de Dados e Algoritmo II

Prof. Romário Nzenguele da Silva

Licenciatura em Informática

2024 - 2025

Sobre o Documento

Material de apoio **gratuito**, desenvolvido exclusivamente para os estudantes do **3º Ano da Licenciatura em Informática**;

O conteúdo, sempre que necessário, será atualizado para refletir as mudanças e melhorias no decorrer do curso. Cada atualização será devidamente versionada, permitindo o fácil acompanhamento das revisões;

Este recurso é um **resumo das aulas**, essencial para apoiar no aprendizado da disciplina.

O **repositório da disciplina** (contendo exercícios, conteúdos detalhados, ...) será reencaminhado por email;

Sugestões de melhoria podem ser apresentadas pelo email que se encontra no rodapé.

Conteúdo

1. Revisão de Conceitos básicos de Programação (Java)

1.1. Variáveis e Tipos de Dados
1.2. Entrada e Saída
1.3. Comentários
1.4. Operadores
1.5. Estruturas de Controle de Fluxo
1.6. Métodos
1.7. Estruturas de Dados

Objectivos

- ✓ Utilizar variáveis e estruturas de dados para armazenamento
- ✓ Realizar entrada e saída de dados
- ✓ Comentar códigos e aplicar operadores
- ✓ Implementar Estruturas de Controle de Fluxo
- ✓ Criar e utilizar métodos

1. Revisão de Conceitos básicos de Programação (Java)

Fundamental para os próximos temas...

1.1. Variáveis e Tipos de Dados

- **Variáveis** são utilizadas para armazenar dados.
- **Tipo de dados** -> natureza dos valores que uma variável pode conter.

  Ex: int, float, double, char, String, boolean, ...

  ❖ **Declaração de variável (Sintaxe):**

  <Tipo_de_dados> <nome_variavel>**;** //sem inicialização
  <Tipo_de_dados> <nome_variavel> **=** <valor>**;**

```
String nome = "Oséas";
int idade = 3;
char letra = 'O';
float altura = 1.78F;
double pi = 3.1415926535897 93;
boolean status = true;
```

[Documentação] Para mais detalhes sobre variáveis e tipos de dados (Java), acessar:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

1.2 Entrada e Saída

Java possui a classe System, que fornece métodos para leitura e escrita de dados:

- ✓ **Entrada de Dados**: Para a entrada, Java utiliza a classe **Scanner**, do pacote **java.util**, que permite ler dados do console.
- ✓ **Saída de Dados**: Para saída, utiliza **System.out.print** e **System.out.println,** onde o segundo insere uma nova linha no final.

```java
import java.time.LocalDate;
import java.util.Scanner;

public class App {
    public static void main(String[] args) {

        int anoNascimento;
        int anoActual = LocalDate.now().getYear(); //buscando pelo ano actual de forma dinamica

        Scanner sc = new Scanner(System.in);

        System.out.print("Digitar ano de nascimento: "); //saida
        anoNascimento = sc.nextInt(); //entrada

        System.out.println("Ate dezembro, teras: "+ (anoActual - anoNascimento) +" anos"); //saida
    }
}
```

1.3 Comentários

São utilizados para descrever o código, obedecendo as seguintes formas:

- **Linha única**: //
- **Multilinha**: /* ... */

```
public class App {

    public static void main(String[] args) {

        // unica linha
        /*
           multiplas linhas
           multiplas linhas
           multiplas linhas
         */

    }
}
```

1.4 Operadores

Operadores são símbolos usados para realizar operações sobre variáveis e valores. Eles podem ser classificados em várias categorias com base na funcionalidade que fornecem.

[Documentação] Para mais detalhes sobre operadores (Java), acessar:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Testa, atribuindo diferentes valores para a e b.

```java
// Operadores aritméticos
System.out.println("Adição: " + (a + b));
System.out.println("Subtração: " + (a - b));
System.out.println("Multiplicação: " + (a * b));

// Operador relacional
System.out.println("a é maior que b? " + (a > b));

// Operador lógico
System.out.println("a é menor que b E a é positivo? " + ((a < b) && (a > 0)));

// Operador ternário
int maior = (a > b) ? a : b;
System.out.println("Maior valor: " + maior);

// Operador bitwise
System.out.println("AND bit a bit: " + (a & b));
```

1.5. Estruturas de Controle de Fluxo

1.5.1. Condicionais

- Fundamentais em qualquer linguagem de programação. Permitem a execução de diferentes ações dependendo de determinadas condições.
- Dentre elas, temos:
    - **if:** executa um bloco de código se uma condição for verdadeira.
    - **else:** executa um bloco de código se a condição(no **if**) for falsa.
    - **else if:** permite testar várias possibilidades.
    - **switch:** alternativa ao if-else. Utilizadas quando várias alternativas são associadas ao valor de uma variável.

1.5. Estruturas de Controle de Fluxo

1.5.1. Condicionais: if, if-else

❑ **Sintaxe:**

Bloco if

```
if (<condição>){       //se verdadeiro
    <instruções>;     //faça isso
}
```

Bloco if-else

```
if (<condição>){          //se verdadeiro
    <instruções>;         //faça isso
} else {                  //se falso
   <instruções>;          //faça isso
}
```

```java
// if
if (x > 5) {
    System.out.println("x é maior que 5");
}

// if-else
if (x > 5) {
    System.out.println("x é maior que 5");
} else {
    System.out.println("x não é maior que 5");
}
```

1.5. Estruturas de Controle de Fluxo

1.5.1. Condicionais: else if

❑ **Sintaxe:**

Bloco else if

```
if (<condição>){          //se verdadeiro
    <instruções>;         //faça isso
} else if (<condição>){   //senão, testa nova condição. Se verdadeiro
    <instruções>;         //faça isso
} else {                  //se falso
   <instruções>;          //faça isso
}
```

```java
// else if
if (x > 5) {
    System.out.println("x é maior que 5");
} else if (x == 5) {
    System.out.println("x é igual a 5");
} else {
    System.out.println("x é menor que 5");
}

// else if
if (nota >= 10) {
    System.out.println("Aprovado");
} else if (nota >= 7) {
    System.out.println("Em recuperação");
} else {
    System.out.println("Reprovado");
}
```

1.5. Estruturas de Controle de Fluxo

1.5.1. Condicionais: switch

❑ **Sintaxe:**

Bloco swicth

```
switch (<variável>){
    case <valor>:
        <instruções>;
        break;
    default:
        <instruções>;
        break;
}
```

Nota: O numero de **cases** depende das opções a serem implementadas. Cada case representa uma condição.

```
switch (dia) {
    case 1:
        System.out.println("Segunda-feira");
        break;
    case 2:
        System.out.println("Terça-feira");
        break;
    case 3:
        System.out.println("Quarta-feira");
        break;
    case 4:
        System.out.println("Quinta-feira");
        break;
    case 5:
        System.out.println("Sexta-feira");
        break;
    default:
        System.out.println("Número inválido. Inserir números entre 1 e 5.");
        break;
}
```

1.5. Estruturas de Controle de Fluxo

1.5.1. Laços de repetição (loops)

- Assim como as condicionais, são fundamentais em qualquer linguagem de programação.
- Permitem a execução de ciclos ou laços de repetição, obedecendo a uma determinada condição.

- O Java suporta os três laços principais:

for

(melhor para)

- ✓ Sequências (contadores) que seguem um padrão específico;
- ✓ Percorrer arrays/vetores;
- ✓ Quando sabes exatamente a quantidade de iterações necessárias.

while

(melhor quando)

- ✓ A condição de parada depende de um evento externo (entrada do usuário, arquivo, etc);
- ✓ A quantidade de iterações necessárias é desconhecida.

do while

(melhor quando)

- ✓ Precisamos garantir que o código execute pelo menos uma vez;

1.5. Estruturas de Controle de Fluxo

1.5.1. Laços de repetição (loops)

- **Sintaxe:**

Laço for

for (<inicialização>; <condição>; <incremento/decremento>){

    <instruções>;

}

Laço while

<inicialização>

while (<condição>){

    <instruções>;

    <incremento/decremento>;

}

```java
// Imprimir os números de 0 a 4
for (int i = 0; i < 5; i++) {
    System.out.println("Número: " + i);
}

int contador = 0;

// Imprimir os números de 0 a 4
while (contador < 5) {
    System.out.println("Número: " + contador);
    contador++;
}
```

1.5. Estruturas de Controle de Fluxo

1.5.1. Laços de repetição (loops)

- **Sintaxe:**

Laço do while

<inicialização>

do{

<instruções>**;**

<incremento/decremento>**;**

} while (<condição>)**;**

```
int contador = 0;

do {
    System.out.println("Número: " + contador);
    contador++;
} while (contador < 5);
```

1.6. Métodos/Funções

- **Blocos de código que executam instruções específicas.**
  - São definidos uma vez e podem ser chamados sempre que necessário, ou seja, permitem a reutilização do código.

1.6.1. Métodos sem retorno

- **Sintaxe:**

```
//sem parâmetros
void <nome>(){
        <instruções>;
}
```

```
//com parâmetros
void <nome>(<parâmetros>){
        <instruções>;
}
```

1.6.2. Métodos com retorno

- **Sintaxe:**

```
//sem parâmetros
<tipo_dados>  <nome>(){
        <instruções>;
        return <valor>;
}
```

```
//com parâmetros
void <nome>(<parâmetros>){
        <instruções>;
}
```

1.6.3. Chamada (utilização) de métodos

```
//sintaxe
<nome>();
<nome>(<argumentos>);
```

Métodos sem retorno

```java
public class App {
    // Método sem argumentos
    1 usage
    static void imprimirMensagem() {
        System.out.println("Hello world!");
    }

    // Método com argumentos
    1 usage
    static void somarEImprimir(int a, int b) {
        int resultado = a + b;
        System.out.println("Soma: " + resultado);
    }

    public static void main(String[] args) {
        // Executando os métodos
        imprimirMensagem();
        somarEImprimir( a: 5,  b: 7);
    }
}
```

Métodos com retorno

```java
public class App {
    // sem argumentos
    1 usage
    public static String imprimirMensagem() {
        return "Hello world!";
    }

    // com argumentos
    1 usage
    public static int somar(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        // executando os métodos
        String msg = imprimirMensagem();
        int soma = somar( a: 5,  b: 7);

        System.out.println("\nSoma = " + soma);

    }
}
```

Nota: o retorno pode ser atribuído a uma variável do mesmo tipo.

1.6. Métodos/Funções

1.6.4 Escopo de Variáveis

O escopo de uma variável define onde ela é acessível.

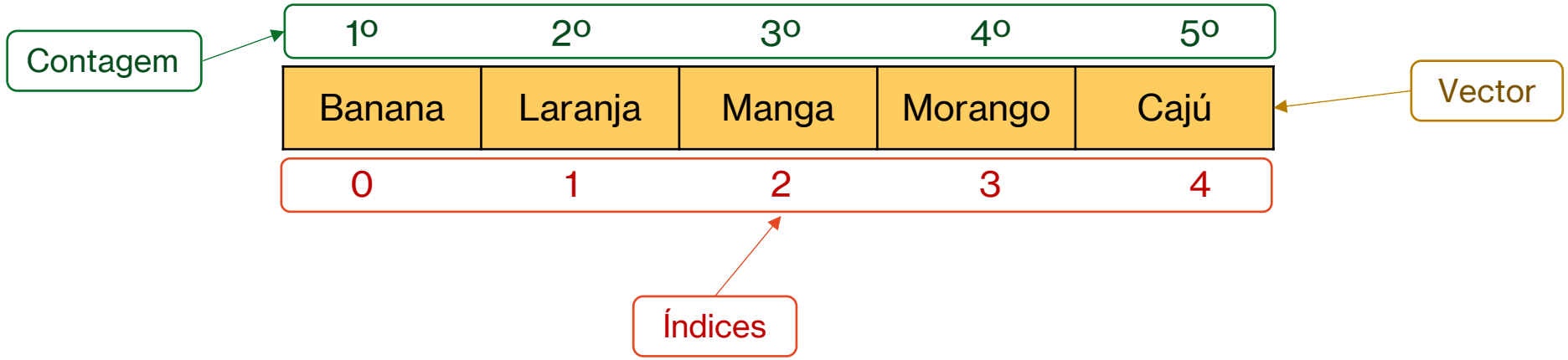
- ✓ **Locais**: são acessíveis apenas dentro do método onde foram declaradas;
- ✓ **Globais**: são acessíveis em qualquer parte do programa.

```java
public class App {

    // Variável global
    3 usages
    static int soma = 0; // Esta variável é acessível em qualquer parte do o programa
    1 usage
    static void calcularSoma() {
        // Variáveis Locais: não podem ser acessadas fora deste método
        int a = 5; // variável Local
        int b = 10; // variável Local

        // Acessando a variável global soma
        soma = a + b;
        System.out.println("a + b = " + soma);
    }

    public static void main(String[] args) {
        calcularSoma();
        // Acessando a variável global soma
        System.out.println("O dobro da soma = " + 2*soma);
    }
}
```

1.7. Estruturas de Dados

1.7.1 Vetores

- Um vetor é uma coleção de elementos do mesmo tipo;
- É acessado por um índice. Os índices começam em 0.

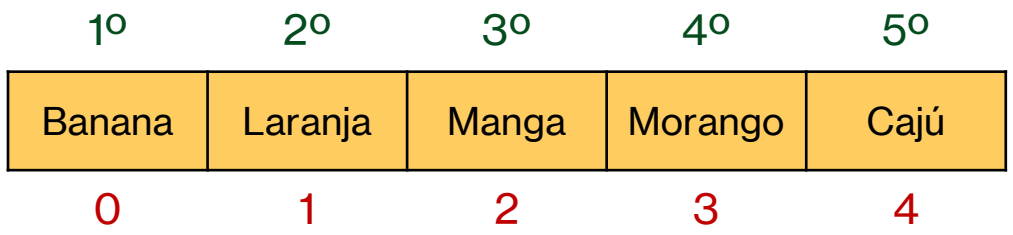


1.7. Estruturas de Dados

1.7.1 Vetores

- **Sintaxe:**

```
//declaração
<tipo_dados>[ ] <nome_vector>= {<valores>};

//acesso
 <nome_vector>[<índice>];
```




```java
public class App {

    public static void main(String[] args) {
        // Primeira forma - declaração e inicialização em uma linha
        String[] frutas = {"Banana", "Laranja", "Manga", "Morango", "Caju"};

        // exibindo a terceira fruta
        System.out.println(frutas[2]);

        // Segunda forma - declaração e inicialização separadas
        String[] frutas2 = new String[5];
        frutas2[0] = "Banana";
        frutas2[1] = "Laranja";
        frutas2[2] = "Manga";
        frutas2[3] = "Morango";
        frutas2[4] = "Caju";

        // exibindo a terceira fruta
        System.out.println(frutas2[2]);
    }
}
```

1.7. Estruturas de Dados

1.7.1 Vetores

- **Percorrendo um vector:**

  Para percorrer um vector, precisamos de utilizar uma estrutura de repetição.

```java
public class App {

    public static void main(String[] args) {
        // Declaração e inicialização do vector
        String[] frutas = {"Banana", "Laranja", "Manga", "Morango", "Caju"};

        //Percorrendo o vector
        for (int i = 0; i < frutas.length; i++) {
            System.out.println(frutas[i]);
        }

    }
}
```

1.7. Estruturas de Dados

1.7.2 Matrizes

- Coleção de elementos apresentados em forma de linhas (i) e colunas (j);
- É um vector de vectores;
- O acesso é feito pelos índices da coluna e linha. Os índices começam em 0.

0	1	2	
Jogador	**País**	**Clube**	0
Vini jr.	Brasil	Real Madrid	1
Lamine Yamal	Espanha	Barcelona	2
Mike Maignan	França	AC Milan	3
Romário Faria	Brasil	Aposentado	4
Toni Kroos	Alemanha	Aposentado	5

j = índice-colunas

i = índice-linhas

1.7. Estruturas de Dados

1.7.2 Matrizes

- **Sintaxe:**

```
//declaração
<tipo_dados> <nome_matriz> [<tamanho_linha>] [<tamanho_lcoluna>]={{<valores>}};

//acesso
 <nome_matriz >[<índice_linha>] [<índice_coluna>];
```

```java
// Declaração e inicialização do vector
String[][] jogadores = {
        {"Jogador", "Pais", "Clube"},
        {"Vini jr.", "Brasil", "Real Madrid"},
        {"Lamine Yamal", "Espanha", "Barcelona"},
        {"Mike Maignan", "França", "AC Milan"},
        {"Romário Faria", "Brasil", "Aposentado"},
        {"Toni Kroos", "Alemanha", "Aposentado"}
};

System.out.println(jogadores[3][1]); // i = 3, j = 1. Saída: "França"
```

1.7. Estruturas de Dados

1.7.2 Matrizes

Percorrendo uma matriz:

Para percorrer uma matriz, utilizamos 2 estruturas de repetição. Uma para iterar as linhas (i) e a outra para iterar as colunas (j).

```java
// Declaração e inicialização do vector
String[][] jogadores = {
        {"Jogador", "Pais", "Clube"},
        {"Vini jr.", "Brasil", "Real Madrid"},
        {"Lamine Yamal", "Espanha", "Barcelona"},
        {"Mike Maignan", "França", "AC Milan"},
        {"Romário Faria", "Brasil", "Aposentado"},
        {"Toni Kroos", "Alemanha", "Aposentado"}
};

//Calculando tamanho da matriz
int linhas = jogadores.length;
int colunas = jogadores[0].length;

//percorrendo a matriz
for (int i = 0; i < linhas; i++) {
    for (int j = 0; j < colunas; j++) {
        System.out.print(jogadores[i][j] + "\t");
    }
    System.out.println();
}
```

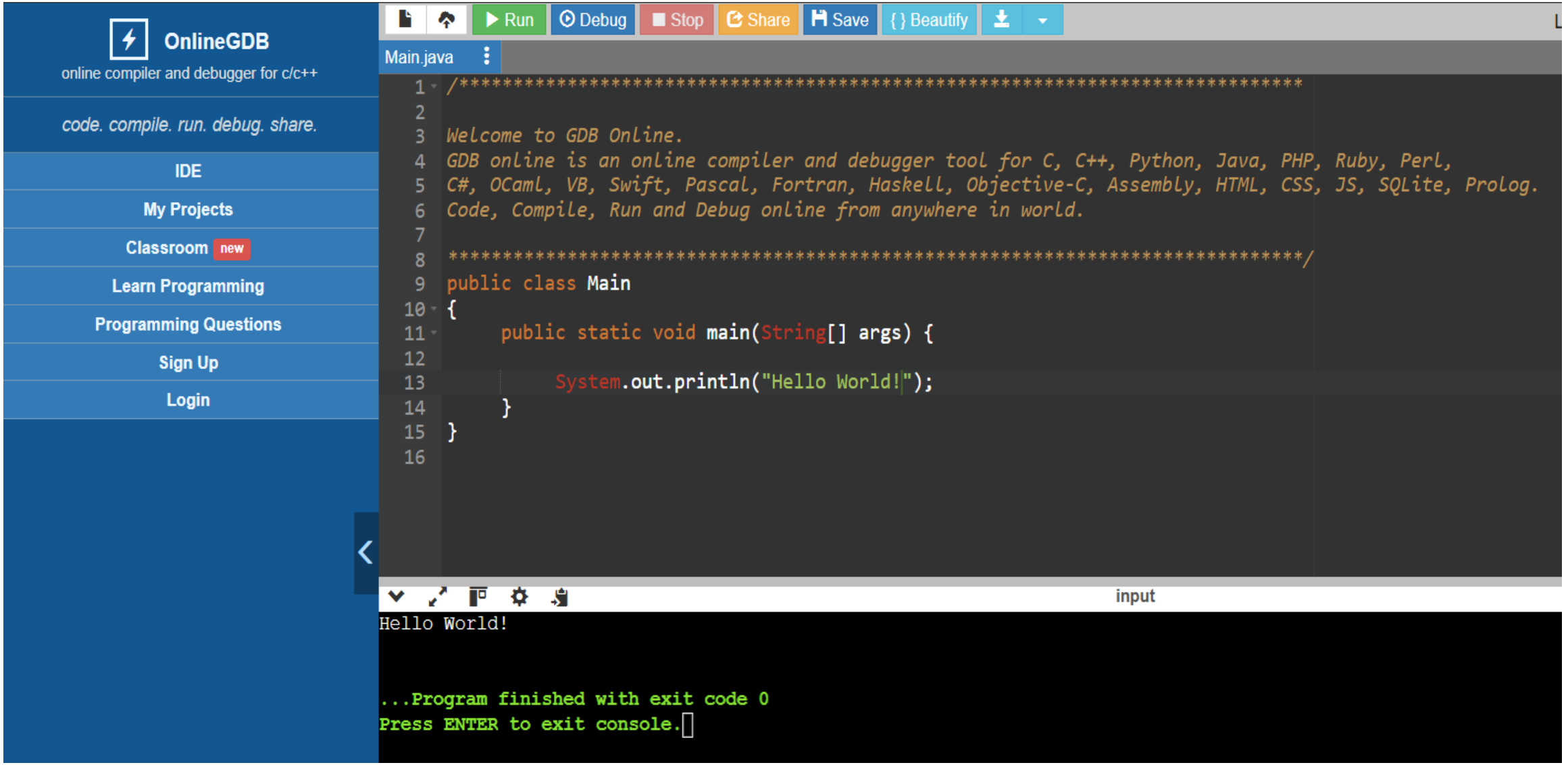
ANEXOS

Ferramentas de Desenvolvimento

- **Intellij IDEA:** https://www.jetbrains.com/idea/
- VS Code: https://code.visualstudio.com/docs/languages/java
- Eclipse: https://eclipseide.org/

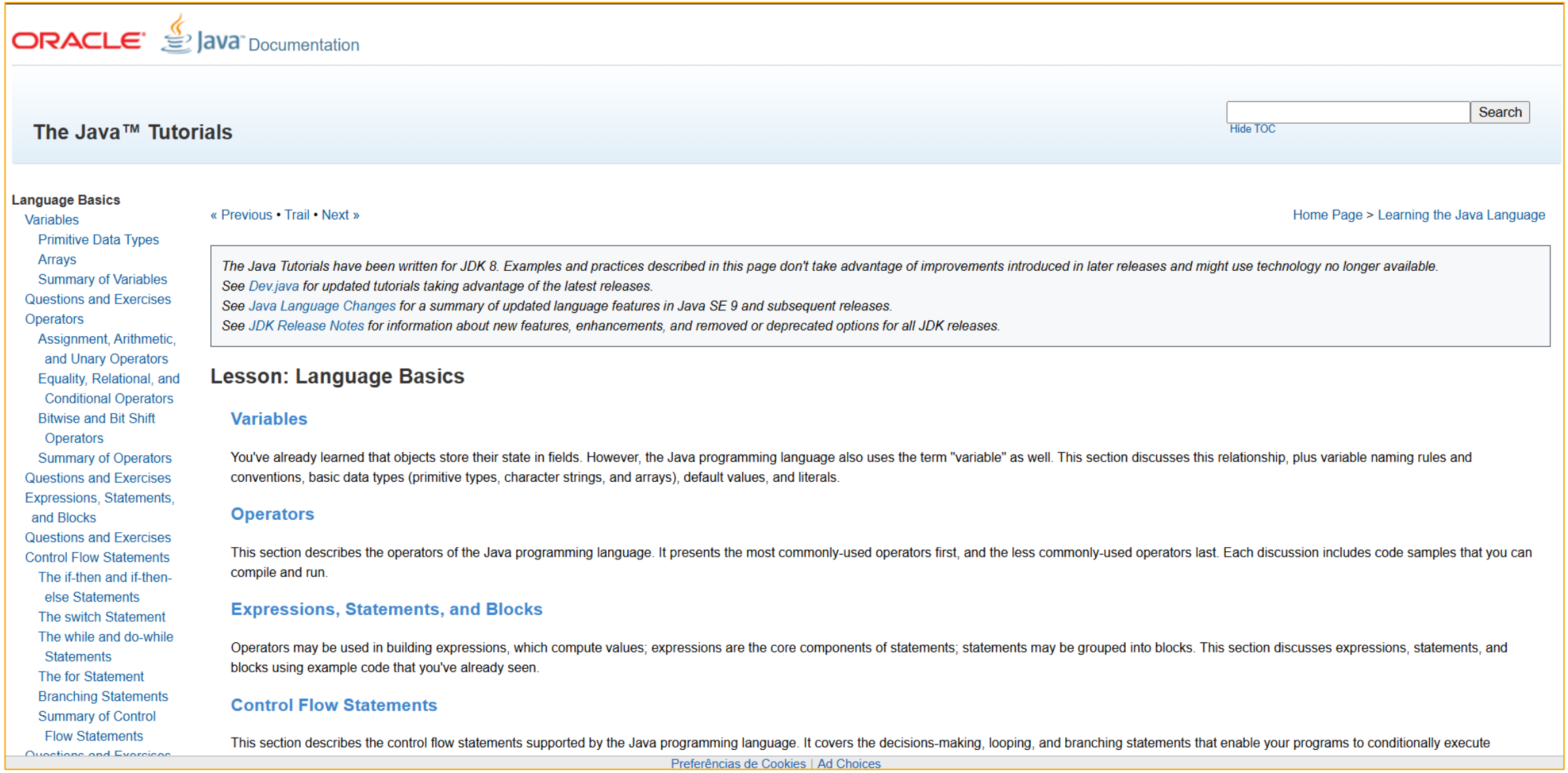
Ferramenta de desenvolvimento (online)

(https://www.onlinegdb.com/)



Documentação oficial (Java)

(https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html)



ORACLE Java Documentation

The Java™ Tutorials

Search

Hide TOC

Language Basics

- Variables
  - Primitive Data Types
  - Arrays
  - Summary of Variables
- Questions and Exercises
- Operators
  - Assignment, Arithmetic, and Unary Operators
  - Equality, Relational, and Conditional Operators
  - Bitwise and Bit Shift Operators
  - Summary of Operators
- Questions and Exercises
- Expressions, Statements, and Blocks
- Questions and Exercises
- Control Flow Statements
  - The if-then and if-then-else Statements
  - The switch Statement
  - The while and do-while Statements
  - The for Statement
  - Branching Statements
  - Summary of Control Flow Statements
- Questions and Exercises

« Previous • Trail • Next »

Home Page > Learning the Java Language

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Dev.java for updated tutorials taking advantage of the latest releases.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Lesson: Language Basics

Variables

You've already learned that objects store their state in fields. However, the Java programming language also uses the term "variable" as well. This section discusses this relationship, plus variable naming rules and conventions, basic data types (primitive types, character strings, and arrays), default values, and literals.

Operators

This section describes the operators of the Java programming language. It presents the most commonly-used operators first, and the less commonly-used operators last. Each discussion includes code samples that you can compile and run.

Expressions, Statements, and Blocks

Operators may be used in building expressions, which compute values; expressions are the core components of statements; statements may be grouped into blocks. This section discusses expressions, statements, and blocks using example code that you've already seen.

Control Flow Statements

This section describes the control flow statements supported by the Java programming language. It covers the decisions-making, looping, and branching statements that enable your programs to conditionally execute

Preferências de Cookies | Ad Choices

...

Sequência -> **M02**