

Linguagem de Programação I

Aula 2

Fundamentos da Linguagem de Programação Orientada a Objetos

`l.bertholdo@ifsp.edu.br`

Conteúdo

- Introdução à Linguagem Java
- Tipos de Dados Primitivos
- Operadores
- Vetores e Matrizes
- Estruturas Condicionais
- Estruturas de Repetição

Introdução à Linguagem Java

- Linguagem de programação compilada e interpretada, de alto nível e multiplataforma.
- Foi lançada pela empresa Sun Microsystems em 1995, sendo mantida atualmente pela Oracle Corporation.
- É uma linguagem orientada a objetos e fortemente tipada, assim como C, C++, C#.
- Permite o desenvolvimento de aplicações nas plataformas *desktop*, *web* e *mobile*.

Tipos de Dados Primitivos

- Representam o tipo primitivo de informação que as variáveis ou constantes podem armazenar.

Tipos de Dados	Tamanho (bytes)	Dados Armazenados
byte	1	Números inteiros com 8 bits de precisão (-128 a 127).
short	2	Números inteiros com 16 bits de precisão (-32.768 a 32.767).
int	4	Números inteiros com 32 bits de precisão (~ -2.15 bilhões a ~ 2.15 bilhões).
long	8	Números inteiros com 64 bits de precisão (~ -9.22 quintilhões a ~ 9.22 quintilhões).
float	4	Números reais com 32 bits de precisão (até 7 casas decimais).
double	8	Números reais com 64 bits de precisão (até 15 casas decimais).
char	2	Caracteres representados em 16 bits (0 a 65.535, se usado como inteiro).
boolean	1	Valor booleano (true ou false).

* Em Java, não existe um tipo de dado primitivo para representar cadeias de caracteres. Para isso, como tipo de dado é utilizada uma classe da API Java denominada **String**.

Operadores Matemáticos

- Operadores para realização de cálculos matemáticos com dados numéricos.

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão
++	Incremento
--	Decremento

```
double a = 5, b = 2;
double resultado;

resultado = a + b;
System.out.println(resultado);
resultado = a - b;
System.out.println(resultado);
resultado = a * b;
System.out.println(resultado);
resultado = a / b;
System.out.println(resultado);
resultado = a % b;
System.out.println(resultado);

a++; // Equivale a: a = a + 1;
System.out.println(a);
a--; // Equivale a: a = a - 1;
System.out.println(a);
```

Operadores de Atribuição

- Operadores usados para atribuir um valor a uma variável.

Exemplos:

```
int numeroRegistros = 25;
```

```
double mediaSalarial = totalSalarios / totalFuncionarios;
```

```
double areaTriangulo = Calcular();
```

Operador	Descrição
+=	Adição com Atribuição
-=	Subtração com Atribuição
*=	Multiplicação com Atribuição
/=	Divisão com Atribuição
%=	Resto da Divisão com Atribuição

```
double x = 8;

x += 5; // Equivale a: x = x + 5;
System.out.println(x);
x -= 3; // Equivale a: x = x - 3;
System.out.println(x);
x *= 4; // Equivale a: x = x * 4;
System.out.println(x);
x /= 2; // Equivale a: x = x / 2;
System.out.println(x);
x %= 3; // Equivale a: x = x % 3;
System.out.println(x);
```

Operadores de Comparação

- Operadores para realização de comparações entre os valores de duas variáveis ou de uma variável e um número.

Operador	Descrição
var1 == var2	Verdadeiro se var1 for igual à var2.
var1 != var2	Verdadeiro se var1 for diferente de var2.
var1 > var2	Verdadeiro se var1 for maior que var2.
var1 < var2	Verdadeiro se var1 for menor que a var2.
var1 >= var2	Verdadeiro se var1 for maior ou igual à var2.
var1 <= var2	Verdadeiro se var1 for menor ou igual à var2.

Operadores de Comparação

```
int x = 5, y = 2;

if (x == y)
    System.out.println("Os operandos são iguais.");
else
    System.out.println("Os operandos são diferentes.");

if (x != y)
    System.out.println("Os operandos são diferentes.");
else
    System.out.println("Os operandos são iguais.");

if (x > y)
    System.out.println("O 1º operando é maior que o 2º operando.");
else
    System.out.println("O 1º operando é menor ou igual que o 2º operando.");
```


Operadores Lógicos

- Operadores usados na construção de expressões condicionais.

Operador	Descrição
! exp1	Verdadeiro se exp1 for falsa.
exp1 && exp2	Verdadeiro se exp1 e exp2 forem verdadeiras.
exp1 exp2	Verdadeiro se exp1 ou exp2 forem verdadeiras.

Operadores Lógicos

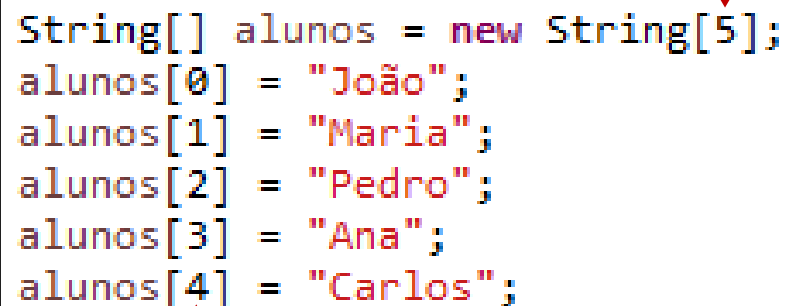
```
int media = 7, presenca = 80;
if ((media >= 6) && (presenca >= 75))
    System.out.println("O aluno foi aprovado.");
else
    System.out.println("O aluno foi reprovado.");

String nota = "ótimo";
if ((nota == "ótimo") || (nota == "bom"))
    System.out.println("O aluno foi aprovado.");
else
    System.out.println("O aluno foi reprovado.");

String situacao = "reprovado";
if (!(situacao == "aprovado"))
    System.out.println("O aluno foi reprovado.");
else
    System.out.println("O aluno foi aprovado.");
```

Vetores

- Vetores ou *arrays* unidimensionais são variáveis que permitem armazenar mais de um valor ao mesmo tempo.
- Cada valor é armazenado em uma posição do vetor, a qual é referenciada por um **índice**.



```
String[] alunos = new String[5];
alunos[0] = "João";
alunos[1] = "Maria";
alunos[2] = "Pedro";
alunos[3] = "Ana";
alunos[4] = "Carlos";
```

tamanho

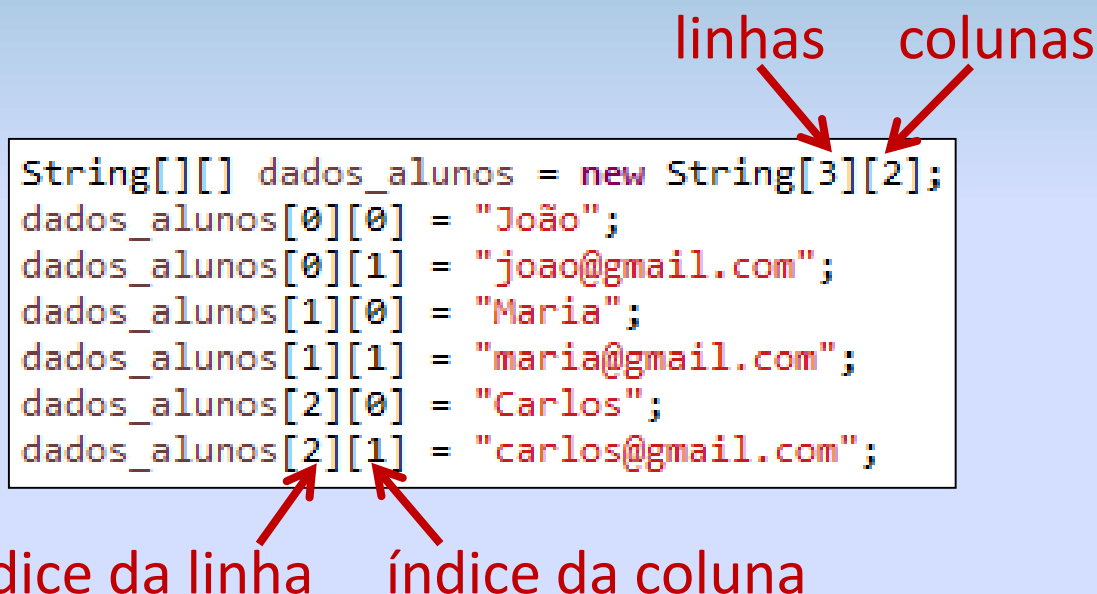
índice

Outra forma de inicialização do vetor:

```
String[] alunos = {"João", "Maria", "Pedro", "Ana", "Carlos"};
```

Matrizes

- Matrizes ou *arrays* bidimensionais permitem armazenar de forma tabular mais de um valor ao mesmo tempo.
- Porém, cada posição do vetor é referenciada por dois **índices**, um representando as **linhas** e outro as **colunas** da matriz.



The diagram illustrates the initialization of a 2D array in Java. A code block shows the declaration and assignment of a `String[][]` array named `dados_alunos`. The array is initialized with 3 rows and 2 columns. Red arrows point from the labels `linhas` and `colunas` to the dimensions `3` and `2` in the array declaration. Another set of red arrows points from the labels `índice da linha` and `índice da coluna` to the indices `0` and `1` in the first element access `dados_alunos[0][0]`.

```
String[][] dados_alunos = new String[3][2];
dados_alunos[0][0] = "João";
dados_alunos[0][1] = "joao@gmail.com";
dados_alunos[1][0] = "Maria";
dados_alunos[1][1] = "maria@gmail.com";
dados_alunos[2][0] = "Carlos";
dados_alunos[2][1] = "carlos@gmail.com";
```

linhas colunas

índice da linha índice da coluna

Outra forma de inicialização da matriz:

```
String[][] dados_alunos = { {"João", "joao@gmail.com"},
                             {"Maria", "maria@gmail.com"},
                             {"Carlos", "carlos@gmail.com"} };
```

Comandos Condicionais

- Permitem avaliar uma expressão e, conforme o resultado obtido, executar um determinado trecho de código.
- São comandos usados em tomadas de decisão dentro do código.

Comandos Condicionais – IF

- O comando **if** pode possuir como complemento o **else if** (senão se) e/ou o **else** (senão). Se todos os blocos anteriores forem falsos, o bloco **else** é executado.
- Somente um dos blocos (if, else if ou else) pode ser processado em cada execução. Ao processar um bloco, todos os blocos seguintes são ignorados.
- Se houver mais de uma linha de código dentro de um bloco (if, else if ou else), é preciso usar chaves (“{ }”) no início e no fim do bloco.

*Um bloco **if** pode ser composto por:*

- *Um único **if**; ou*
- *Um único **if** e um único **else**; ou*
- *Um único **if** e uma ou mais instruções **else if**; ou*
- *Um único **if**, uma ou mais instruções **else if** e um único **else**.*

Comandos Condicionais – IF

```
double prova1 = 7;
double prova2 = 5;
double media = 0;
String desempenho;

media = (prova1 + prova2) / 2;

if (media <= 5)
    desempenho = "INSATISFATÓRIO";
else if (media <= 7)
    desempenho = "REGULAR";
else if (media <= 8.5)
    desempenho = "BOM";
else
    desempenho = "ÓTIMO";

System.out.println("O desempenho do aluno foi " + desempenho);
```

Comandos Condicionais – SWITCH

- Usado para verificar se o conteúdo de uma variável é **igual** a um valor dentre vários possíveis.
- O comando **break** encerra o bloco **switch**, de modo que a execução do programa continue após o bloco.
- A opção **default** tem a mesma função da opção **else** do comando **if**. Se todos os blocos anteriores forem falsos, o bloco **default** é executado.
- Para testar mais de um valor e executar o mesmo código, basta declarar os testes desejados e incluir o código a ser executado após o último valor testado.

Comandos Condicionais – SWITCH

```
String opcao = "sim";

switch (opcao) {
    case "sim":
        System.out.println("Você escolheu a opção SIM");
        break;
    case "não":
    case "talvez":
        System.out.println("Você não escolheu a opção SIM");
        break;
    default:
        System.out.println("A opção digitada é inválida");
        break;
}
```

Comandos de Repetição

- Permitem executar um trecho de código repetidamente por um determinado número de vezes, ou até que uma condição seja satisfeita.

Comandos de Repetição – WHILE

- Avalia uma expressão e, enquanto a expressão for verdadeira, a execução do bloco de comandos é repetida. Quando a expressão for falsa, o laço (*loop*) de repetição é encerrado e a execução do programa continua após o final do bloco **while**.
- Se houver mais de uma linha de código dentro de um bloco **while**, é preciso usar chaves (“{ }”) no início e no fim do bloco.

```
int cont = 1;

while (cont < 5) {
    System.out.println("O valor do contador é " + cont);
    cont++;
}
```

Comandos de Repetição – DO/WHILE

- Semelhante ao comando **while**. Porém, como a expressão **while** é avaliada somente no final do laço, o bloco de comandos é executado pelo menos uma vez.

```
int cont = 1;

do {
    System.out.println("O valor do contador é " + cont);
    cont++;
} while (cont < 5);
```

Comandos de Repetição – FOR

- Usado quando se deseja executar um trecho de código um determinado número de vezes. **Exemplo:** Imprimir todos ou parte dos elementos de um array.
- Se houver mais de uma linha de código dentro de um bloco **for**, é preciso usar chaves (“{ }”) no início e no fim do bloco.

Comandos de Repetição – FOR

- Laço de repetição controlado por índice
 - Nesse caso, o comando **for** utiliza um **índice** para controlar os laços de repetição e, se necessário, acessar apenas parte dos elementos de um array.
 - O **índice** tem um **valor inicial** (condição de início), um **valor final** (condição de parada) e um **valor de incremento** ou **decremento** que aumenta (+1) ou diminui (-1) o **índice** a cada laço executado.

```
for (int i = 0; i < 10; i++)  
    System.out.println("Informe o " + (i + 1) + "º produto: ");  
  
char[] elementos = { 'A', 'B', 'C', 'D', 'E' };  
for (int i = 0; i < 3; i++)  
    System.out.println("O valor do elemento é " + elementos[i]);
```

Comandos de Repetição – FOR

- Laço de repetição não controlado por índice
 - Usado quando se deseja acessar todos os elementos de um array, não sendo necessárias condições de início ou de parada.

```
char[] elementos = { 'A', 'B', 'C', 'D', 'E' };  
  
for (char x : elementos)  
    System.out.println("O valor do elemento é " + x);
```

```
String[][] pessoas = { { "João", "36" }, { "Maria", "28" }, { "Pedro", "45" } };  
  
for (String[] p : pessoas) // Trata cada linha da matriz como um vetor unidimensional.  
    System.out.println(Arrays.toString(p));
```

```
[João, 36]  
[Maria, 28]  
[Pedro, 45]
```

Comandos de Repetição – BREAK

- Usado quando se deseja interromper a execução de um bloco de repetição.
- Quando o programa lê o comando **break**, a execução do bloco é interrompida independentemente da condição de parada do comando de repetição.

```
int i = 1;

while (i < 10) {
    if (i == 5)
        break;

    System.out.print(i + ", ");
    i++;
}
```



1, 2, 3, 4,

Comandos de Repetição – CONTINUE

- Usado quando se deseja ir para o próximo laço de repetição de um bloco, sem que o laço atual tenha chegado ao final.
- Quando o programa lê o comando **continue**, a execução do laço atual é interrompida e um novo laço de repetição é iniciado.

```
int i = 1;

while (i < 10) {
    if (i == 5) {
        i++;
        continue;
    }
    System.out.print(i + ", ");
    i++;
}
```



1, 2, 3, 4, 6, 7, 8, 9,

Referências

- Peter Jandl Junior; Java – Guia do Programador – 3ª Edição. São Paulo: Novatec Editora, 2015.
- Rafael Santos; Introdução à Programação Orientada a Objetos usando Java – 2ª edição. Rio de Janeiro: Elsevier, 2013.