



Análise Orientada a objetos

Aula 3

Prof. Me. Juliana Costa-Silva



Na aula de hoje...

1. Estruturas de repetição
2. Desvios em Repetição
3. Arrays
4. Arrays Multidimensionais
5. Propriedades
6. Manipulação de Strings
7. Atividades



Estrutura de repetição while

Sintaxe:

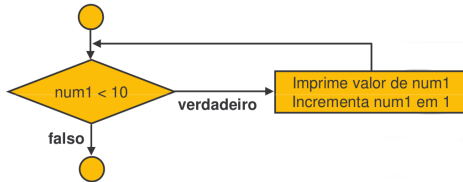
```
while ( condição ) {  
    // bloco de instruções a ser executado enquanto a  
    // condição for verdadeira  
}
```

Em Java, para identificar a **condição** de teste é **obrigatório** o uso de **parênteses**

A expressão a ser avaliada na condição deve retornar um valor booleano (**boolean**)



while (enquanto)



```
1 int num1 = 0;
2 while( num1 < 10){
3     System.out.println("- " +(num1++));
4 }
```



while (enquanto) - Exemplo

```
1  int num2 = 0;
2  // Errado
3  while(num2 < 10 && num2 > -5 && num2 != 10);
4  {
5      num2 = 11;
6      System.out.println(" * " + num2);
7  }
8  // Correto
9  while(num2 < 10 && num2 > -5 && num2 != 10){
10     num2 = 11;
11     System.out.println(" * " + num2);
12 }
```



do/ while (faça/ enquanto)

Enquanto a condição de teste for verdadeira continua repetindo a execução do conjunto definido de instruções.

```
do {  
    // bloco de instruções a ser executado enquanto a  
    // condição for verdadeira  
} while ( condição );
```

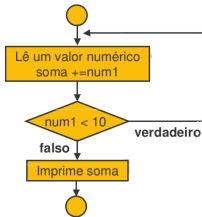
NÃO esquecer do ;

Qual a diferença entre while e do while?

A diferença em relação ao while é que o do/while realiza o teste após executar pelo menos **uma vez** o bloco de instruções.



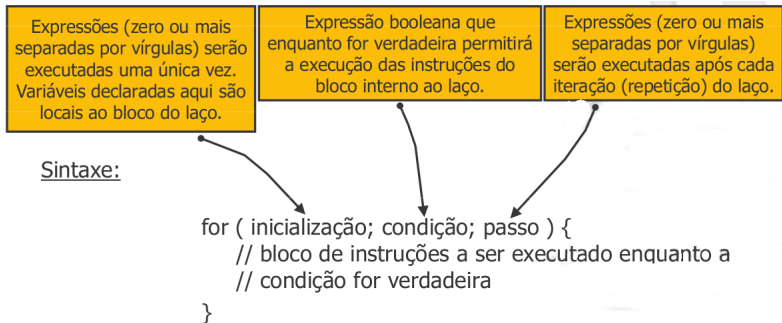
do/ while - Exemplo



```
1 int num3 = 0, soma = 0;
2 do{
3     num3 = Integer.parseInt(JOptionPane.
4         showInputDialog(null, "Valor?"));
5     soma += num3;
6 }while(num3 > 0);
7 System.out.println("Soma total: " + soma);
```

Java for

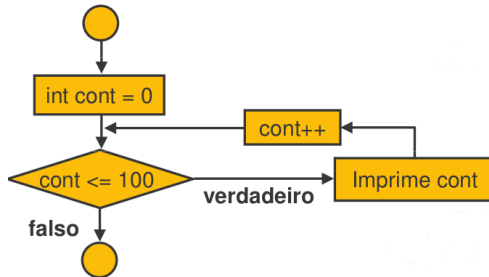
Estrutura para execução de um bloco de instruções por repetidas vezes. A condição testada para a execução. A condição testada para a execução normalmente é baseada em um contador.





for - Exemplo

```
1   for(int cont = 0; cont <= 100; cont++ ){  
2       System.out.println(cont);  
3   }
```





Exemplos de uso do for

1. for (int i = 1; i <= 100; i++)

Conta de 1 a 100, de 1 em 1.

2. for (int i = 100; i >= 100; i--)

Conta de 100 a 1, de 1 em 1.

3. for (int i = 7; i >= 77; i+= 7)

Conta de 7 a 77, de 7 em 7.

4. for (int i = 20; i >= 20; i -=2)

Conta de 20 a 2, de 2 em 2.

5. for (int i = 2; i <= 20; i +=3)

Conta de 2 a 20, de 3 em 3.

6. for (; ;)

Looping infinito.

7. for (int i = 2 ; ; i += 3)

Looping infinito.

8. for (; (i< 10) && j>0);)

Condição composta.



Desvios em repetição

```
while (!terminado) {  
    passePagina();  
    if (alguemChamou == true) {  
        break;           // caia fora deste loop  
    }  
    if (paginaDePropaganda == true) {  
        continue;       // pule esta iteração  
    }  
    leia();  
}  
restoDoPrograma();
```



Desvios em repetição

Comando **break**

- Interrompe a execução de um bloco (**switch**, **while**, **do/ while** e **for**).

Comando **continue**

- Pula as instruções restantes de um bloco (**while**, **do/ while** e **for**), e realiza a próxima iteração do laço.

ATENÇÃO!!

Após o desvio, pelo comando **continue**, nas estruturas **while** e **do/while**, é realizado o teste da condição de repetição.

Na estrutura **for** são executadas as ações de incremento/ decremento e só depois realizado o teste da condição de repetição.



Exemplo **break**

```
1 String nomes[] = {"Joao", "Maria", "Jose"};
2 String procura = "Jose";
3 boolean encontrou = false;
4 for(int i = 0; i<nomes.length; i++){
5     if(nomes[i].equals(procura)){
6         encontrou = true;
7         break;
8     }
9     System.out.println("Visitamos este lugar");
10 }
11 if(encontrou)
12     System.out.println(procura + " encontrado!");
13 else
14     System.out.println(procura + "NAO encontrado");
```



Exemplo **continue**

```
1 String names[] = {"Joao", "Bia", "Pedro", "Joao"};
2 String procura = "Joao";
3 int cont = 0;
4     for (int i = 0; i < names.length; i++) {
5         if(!names[i].equals(procura)){
6             continue;
7         }
8         cont++;
9         System.out.println("Encontrei");
10    }
11 System.out.println(cont+" "+procura+" na lista");
```



Exemplo **continue** 2

```
1 int numero = 0;
2 String saida = "";
3     for (numero = 0; numero < 10; numero++) {
4         if(numero%2 == 0)
5             continue;
6         saida += numero + " ";
7     }
8     JOptionPane.showMessageDialog(null,saida,"Saida",
9         JOptionPane.INFORMATION_MESSAGE);
10    System.exit(0);
```



Classes Wrappers

Java possui algumas classes para auxiliar a utilização de Tipos Primitivos. Estas classes são chamadas de **Wrappers**.

- `java.lang.Boolean;`
- `java.lang.Character;`
- `java.lang.Byte;`
- `java.lang.Integer;`
- `java.lang.Long;`
- `java.lang.Float;`
- `java.lang.Double;`



Classes Wrappers

Cada uma dessas classes trabalha com o tipo de dados que o seu nome indica;

Todos os métodos são do tipo **static** (de classe), ou seja, é possível chamá-los sem a necessidade de se criar um objeto de tipo da classe.



A classe Boolean

A classe Boolean provê métodos para a manipulação de tipos primitivos boolean.

```
1 boolean var1 = Boolean.valueOf("true");  
2 boolean var2 = Boolean.valueOf("False");  
3 boolean var3 = Boolean.valueOf("yes");
```

valueOf

Converte a String em true ou false caso ela sejam um desses textos, independente de maiúsculas e minúsculas.



Byte, Short, Integer e Long

As classes Byte, Short, Integer e Long oferecem maior praticidade na manipulação de tipos primitivos inteiros (byte, short, int e long).

```
1  int inteiro = Integer.parseInt("10");  
2  long longo = Long.parseLong("105");  
3  int min = Integer.MIN_VALUE;  
4  byte bMax = Byte.MAX_VALUE;
```

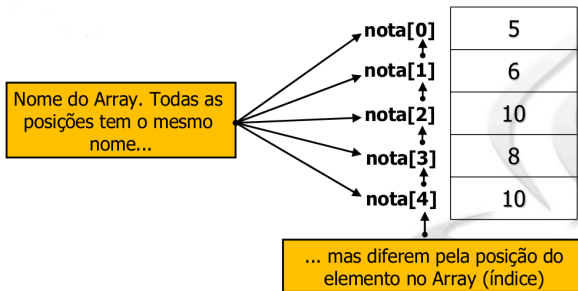


Conversão String — > outros tipos de dados

Descrição	Comando
String → int	<code>int numero = Integer.parseInt("15");</code>
String → double	<code>double d = Double.parseDouble("4.1234567890");</code>
String → float	<code>float f = Float.parseFloat("3.1234567");</code>
String → boolean	<code>boolean b = Boolean.parseBoolean("true");</code>
String → char	<code>char c = varString.charAt(posicaoDoChar);</code>
Todos → String	<code>String s = String.valueOf(2.3F);</code> <code>ou String t = 3.4 + "";</code>



Array



ATENÇÃO:

- o índice obrigatoriamente deve ser um número inteiro;
- a primeira posição do Array é acessada pelo índice zero;
- a última posição tem valor igual ao tamanho do Array menos 1;
- acessar posições fora do Array gera exceção (`ArrayIndexOutOfBoundsException`), que caso não seja tratada encerra a execução da aplicação.



Array

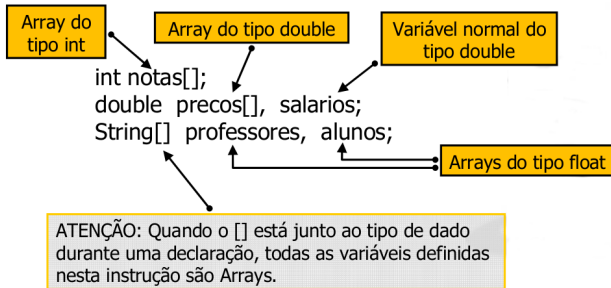
Para se utilizar um Array é necessário realizar os seguintes passos:

- Declarar a variável do tipo Array
- Instanciar o objeto Array (alocar memória)
- Inicializar os valores do objeto Array



Declaração de um Array

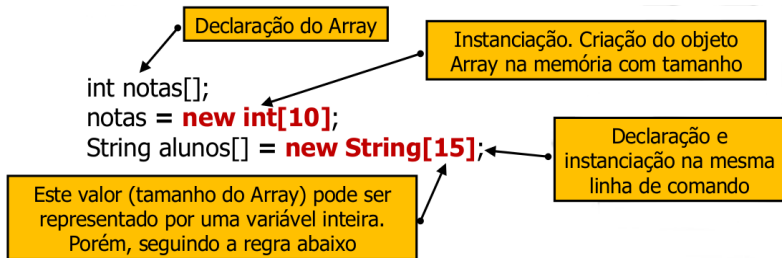
É possível construir Arrays a partir de um tipo primitivo, ou através de uma variável de instância, para referenciar objetos de uma classe.





Instanciação de um Array

A instanciação do novo array é realizada através do operador **new**, como acontece com todos os objetos em Java.





Inicialização de um Array

Após a alocação de um Array todas as sua posição recebem **implicitamente** um valor padrão nulo.

Tipos primitivos

- **byte, short, int** recebem 0 (zero)
- **long** recebe 0L
- **float** recebe 0.0F
- **double** recebe 0.0
- **char** recebe ' 0000'

Tipos construídos

- As variáveis de referência a objetos recebem **null**.



Inicialização de valores de um Array

Após a instanciação de um objeto Array atribuir valores as suas posições utilizando-se o índice de acesso a posição.

```
int notas[] = new int[100];  
notas[0] = 50;  
notas[1] = 70;
```

Após a instanciação do objeto todas as posições recebem o valor padrão **zero**.

```
String ruas[];  
ruas = new String[100];  
ruas[18] = "Higienópolis";  
ruas[19] = "Tietê";
```

Após a instanciação do objeto todas as posições recebem o valor padrão **null**.



Inicialização de valores de um Array

Após a instanciação de um objeto Array atribuir valores as suas posições utilizando-se o índice de acesso a posição.

```
1 String nomes[] = {"Alexandre", "Maria", "Jose"};  
2 double salarios[] = {1500, 754.50, 412.00};  
3 float vet[]={2.5f,3.4f,7.8f,10,9.1f};
```



Array

Utilizando a propriedade de tamanho do Array

```
1  double valores[] = new double[1000];  
2  for (int x = 0; x < valores.length; x++) {  
3      valores[x] = x*2;  
4  }
```

Para receber o tamanho de um array, utiliza a propriedade **length** do tipo int, que representa a quantidade de elementos de um array.



Arrays Multidimensionais

É possível em Java criar vetores com mais de uma dimensão (arrays Multidimensionais).

- Arrays com **uma dimensão** são simplesmente chamados de vetores.
- Arrays com **duas dimensões** ou arrays bidimensionais, normalmente são chamados de **matrizes** e representam uma tabela de valores.
- Para arrays com **N dimensões** são necessários N índices para localizar um elemento, necessitando especificar um índice para cada dimensão definida.



Java

Arrays

Multidimensionais

Para os Arrays multidimensionais são válidas as mesmas regras dos Arrays unidimensionais

- 3 passos (declaração / instanciação / inicialização);
- Quando instanciado, todas suas posições recebem valores nulos;
- Depois de instanciado é imutável (não altera tamanho).



Exemplos

Declaração

Declaração:

```
1 double [][] valores;  
2 String marcaModelo [][];
```

Instanciação:

```
1 valores = new double [2] [4];  
2 marcaModelo = new String [3] [2];
```

Declaração e Instanciação (tudo junto):

```
1 double [][] valores2 = new double [2] [4];  
2 String marcaModelo2 [][] = new String [3] [2];
```



Inicialização

Exemplos

Inicialização:

```
1  valores[1][2] = 3.50;  
2  valores[0][3] = 7.28;  
3  marcaModelo[1][0] = "ford";  
4  marcaModelo[1][1] = "ecosport";
```

Declaração e Instanciação e Inicialização:

```
1  double valores3[][] = {{ 10, 6, 32, 7.28 },  
2                           { 15, 16, 3.50, 44}};  
3  String marcaModelo3[][] =  
4  {{ "gm", "vectra" },  
5   { "ford", "ecosport"},  
6   { "fiat", "stilo"}};
```




Propriedades

Array Multidimensional

Todo Array multidimensional também possui uma propriedade chamada **length** do tipo **int**, para cada uma das dimensões que possui.

Exemplo:

```
double notas[][] = new double[100][4];
for (int linha=0; linha<notas.length; linha++) {
//notas.length:quantidade de linhas
    for(int coluna=0; coluna<notas[linha].length; coluna++) {
//notas[linha].length: quantidade de coluna por linha
        notas[linha][coluna] = 10;
    }
}
```



Strings

- Em Java o tipo **String não é um tipo primitivo** e sim um tipo construído, ou seja, **é uma classe** que encapsula o texto (sequência de caracteres) e os métodos para manipulá-lo.
- A classe String faz parte do pacote **java.lang** e não precisa ser importada explicitamente.
- A classe String integra a API básica do Java, por ser uma classe de uso fundamental na criação de programas.



Strings

- É possível criar objetos da classe String atribuindo diretamente uma constantes String a uma variável de referência do tipo String.
- Outra opção é chamar o construtor da classe String passando a constante String como parâmetro.

Opção 1:

```
1 String nome = "Juliana";
```

Opção 2:

```
1 String nome2 = new String("Juliana");
```




Strings

Tamanho

É possível identificar quantos caracteres estão armazenados num objeto do tipo String acessando o método **length()**. **Exem-**

plo:

```
1 String texto = "teste";  
2 System.out.println(texto.length());
```



Java Strings

Comparação

Não é aconselhável comparar dois objetos String através do operador de igualdade (==), pois, as mesmas só terão o mesmo valor caso ambas tenham sido




Exemplo

Comparação de String

Receba duas Strings e compare e imprima igual ou diferente.

```
import javax.swing.JOptionPane;

public class ExemploString {
    public static void main(String[] args) {
        String nome1 = JOptionPane.showInputDialog(null, "Digite o nome 1");
        String nome2 = JOptionPane.showInputDialog(null, "Digite o nome 2");
        if(nome1 == nome2){
            System.out.println("O objeto referenciado por nome1 [" +
                               nome1 + "] e por nome2 [" +
                               nome2 + "] eh o mesmo!\n");
        }else{
            System.out.println("O objeto referenciado por nome1 [" +
                               nome1 + "] e por nome2 [" +
                               nome2 + "] sao diferentes!\n");
        }
    }
}
```



Java Strings

Comparação

Métodos que comparam Strings

Veja o exemplo em ExemploString2.java

Métodos que retiram partes da String

Veja o exemplo em ExemploString3.java



Atividades

1. Faça um programa em Java que mostre a tabuada de um número escolhido pelo usuário (entre 1 e 10). Repita o exercício com os 3 laços.
2. Faça um programa em Java que mostre a tabuada de 1 a 10 em uma mesma tela. De 1 a 5 no primeiro bloco e do 6 ao 10 no segundo.



Atividades

3. Faça um programa em Java que imprima todos os múltiplos de 3, entre 1 e 100.
4. Faça um programa em Java que calcule o fatorial de um número pré-definido.
5. Escreva um programa em Java que imprima todos os números múltiplos de 5, no intervalo fechado de 1 a 500.



Atividades

6. Escreva uma aplicação capaz de receber 10 números (tipo ponto flutuante), calcule e imprima:

- Os números digitados;
- A soma dos números;
- A média aritmética entre eles;
- O maior número;
- O menor número.

7. Um quadrado mágico é uma matriz quadrada em que a soma das suas linhas é igual a soma das sua colunas e que também é igual a soma da diagonal principal e da diagonal secundária. A matriz abaixo é um exemplo de quadrado mágico, pois a somatória, em todos os casos, é igual a 15.

4	9	2	→15
3	5	7	→15
8	1	6	→15
↙15	↓15	↓15	↘15

Figura 2: A constante mágica no quadrado de Lo Shu

Faça um código Java que receba uma dimensão N de uma matriz $A_{n \times n}$, seguido dos respectivos valores da matriz (preenchendo a matriz da linha 0 até N, da esquerda para a direita: *coluna 0 até N*), verificar se a matriz é um quadrado mágico (Imprima: “quadrado magico” caso seja e “quadrado NÃO magico” caso não seja).



Atividades

8. Escreva uma aplicação que receba do usuário uma frase no seguinte formato N-N-N-...-N-N-N (representando por números inteiros separados por hífen), extraia esses números desta frase e crie e alimente um vetor de tamanho exato a quantidade de números. De posse desses números, coloque-os em ordem decrescente no vetor.



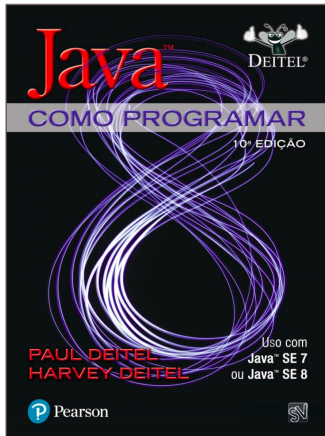
Atividades

9. Faça uma aplicação que receba uma frase e retorne o texto invertido.
10. Faça uma aplicação que receba o nome completo do usuário, e depois troque o seu último sobrenome por Silva. Mostre o resultado na tela.



Leitura complementar

Para mais informações sobre JAVA, leia:



Capítulo 3 a 7: [\[Deitel, 2016\]](#)



Referências



Deitel, Paul J.; Deitel, H. M. (2016).

Java 8: Como programar. 10ª Edição.

Pearson.