



(✓) Curso Técnico

Quem Investe no futuro faz QI

Informática para Internet

Desenvolvimento de Sistemas

Web III

Unidade IV

1. Estrutura de repetição

As estruturas de repetições resolvem o problema relacionado à necessidade de repetir uma mesma sequência de instruções/comandos em um software. A regra é que as repetições sejam condicionadas a um teste lógico ou objetivo, por exemplo: atualizar uma lista de espera enquanto o sistema/software estiver rodando, e parar de atualizar quando ele for finalizado. Outro exemplo seria a execução de N instruções/comandos, guardando a informação de qual instrução/comando está sendo executado no momento. Para o exemplo da atualização, existe o laço **while**. Para o exemplo do armazenamento da informação, existe o laço **for**, observando a aplicação mais adequada das estruturas.

No contexto das estruturas de repetições alguns termos são utilizados, destarte torna-se importante saber o significado:

Loop: mesmo que "laço", se referindo ao fato de que o laço termina no seu início, ou seja, repetição. Na estrutura de repetição os loops são trechos de código que se repetem várias vezes, conforme a definição algorítmica.

Iteração: mesmo que "ciclo". Uma iteração é a execução de um ciclo de um laço, ou seja, se a estrutura está programada para fazer a mesma coisa por dez vezes, cada vez será uma iteração.

2. Estrutura de repetição - for

A sintaxe da estrutura da repetição **for** é a seguinte:

```
for (início ; condição ; passo) {  
    instruções/comandos;  
}
```

Importante explicar detalhadamente cada parte da estrutura de repetição **for**, consoante registra-se:

→ **início** - diretiva executada antes do laço começar possibilitando a atribuição de um valor inicial ao elemento "contador/controlador". Vale salientar que é executado apenas na primeira vez que a estrutura de repetição **for** é acionada;

→ **condição** - é o teste lógico da estrutura de repetição **for** ou a expressão de condição de parada do laço, valendo salientar que sempre é executada na estrutura de repetição **for** acionada;

→ **passo** - diretiva executada após cada iteração do laço, geralmente utilizada para incrementar ou decrementar o elemento "contador/controlador", valendo salientar que será executada após a primeira iteração.

Em síntese o funcionamento da estrutura ocorre da seguinte maneira:

1. Primeiramente, é executado o que está em **início**;
2. Então, repete-se:
 - a. Verifica-se o resultado de **condição**;
 - b. Se for false, interrompe a repetição e segue o código após o **for** ;
 - c. Se for true, executa a sequência de instruções/comandos do **for**;
 - d. Depois de executar a sequência de instruções/comandos do **for**, executa o que estiver no **passo**, e retorna-se ao item a desta lista.

Utilizando a linguagem Java, temos o seguinte exemplo:

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4      <head>
5          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6          <title>Página JSP Estrutura de Repetição</title>
7      </head>
8      <body>
9          <%
10             for (int i = 0; i < 3; i++) {
11                 out.print("Mostrando "+i+"<br>");
12             }
13             out.print("Fim do programa.");
14          %>
15      </body>
16  </html>

```

Mostrando 0
Mostrando 1
Mostrando 2
Fim do programa.

CURSO TÉCNICO EM INFORMÁTICA PARA INTERNET

Vale pontuar o fluxo de execução do exemplo acima:

Nesse caso, o que o programa fará é:

1. O valor de **i** começa como 0;
2. **i** é menor que 3? Ou seja: 0 é menor que 3? Sim, então executa o que estiver no escopo;
3. Mostra o texto "Mostrando 0" (pulando linha);
4. Executa **i++** (ou seja, i agora é 0 + 1, e portanto **i** = 1);
5. **i** é menor que 3? Ou seja: 1 é menor que 3? Sim;
6. Mostra o texto "Mostrando 1";
7. Executa **i++** (agora **i** é 2);
8. **i** é menor que 3? Ou seja: 2 é menor que 3? Sim;
9. Mostra o texto "Mostrando 2";
10. Executa **i++** (agora **i** é 3);
11. **i** é menor que 3? Ou seja: 3 é menor que 3? Não, e portanto saido do **for**;
12. Mostra o texto "Fim do programa.".

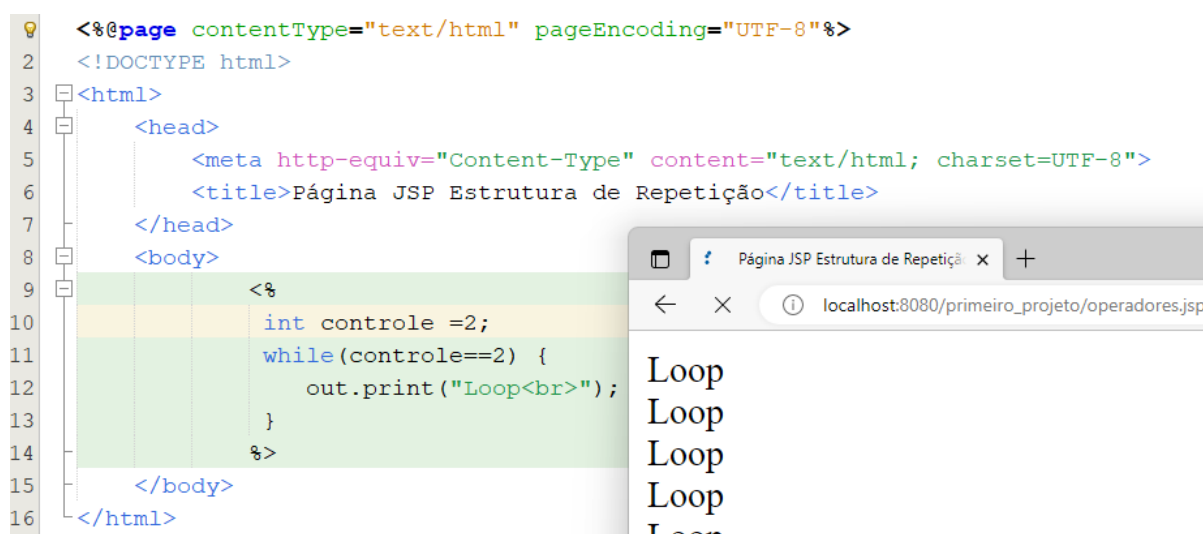
Fluxo	i	Resultado
int i = 0	0	-
i < 3	0	true
out.print("Mostrando "+ i +" ");	0	Mostrando 0
i ++	1	-
i < 3	1	true
out.print("Mostrando "+ i +" ");	1	Mostrando 0
i ++	2	-
i < 3	2	true
out.print("Mostrando "+ i +" ");	2	Mostrando 0
i ++	3	-
i < 3	3	false
out.print("Fim do programa.");	*	Fim do programa

3. Estrutura de repetição - while

A sintaxe da estrutura da repetição **while** é a seguinte:

```
while (condição) {  
  
    instruções/comandos;  
  
}
```

A estrutura de repetição **while** executa uma sequência de comandos até que a condição retorne **false**. Ao analisar o algoritmo abaixo, percebe-se que as repetições não cessarão, pois a condição resultará em **true**, ou seja, na linha 10 a variável controle recebe 2, e na linha 11 a condição estabelecida para execução da repetição é o valor contido na variável controle ser igual a 2:



```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>  
2 <!DOCTYPE html>  
3 <html>  
4 <head>  
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
6 <title>Página JSP Estrutura de Repetição</title>  
7 </head>  
8 <body>  
9 <%  
10     int controle =2;  
11     while(controle==2) {  
12         out.print("Loop<br>");  
13     }  
14 %>  
15 </body>  
16 </html>
```

Página JSP Estrutura de Repetição x +
localhost:8080/primeiro_projeto/operadores.jsp
Loop
Loop
Loop
Loop
Loop

Importante explicar que uma maneira de evitar o loop no código exposto anteriormente é modificar o valor da variável controle dentro da estrutura de repetição **while**, ou utilizar o comando **break**. Ainda sobre o tema, vale lembrar que o comando **break** interrompe as repetições nas estruturas de repetições **while** e **for**. Para melhor entendimento exemplifica-se:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"%>
6     <title>Página JSP Estrutura de Repetição</title>
7 </head>
8 <body>
9     <%
10         int controle =2;
11         while(controle==2) {
12             out.print("Loop<br>");
13             controle=3;
14         }
15     %>
16 </body>
17 </html>

```

Loop

Interrupção utilizando o break:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"%>
6     <title>Página JSP Estrutura de Repetição</title>
7 </head>
8 <body>
9     <%
10         int controle =2;
11         while(controle==2) {
12             out.print("Loop<br>");
13             break;
14         }
15     %>
16 </body>
17 </html>

```

Loop

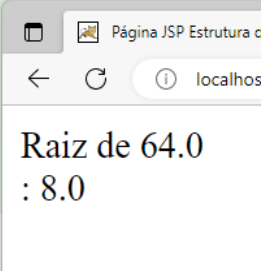
A diferença entre a estrutura de repetição **for** e a **while** é a noção sobre o número de laços de cada estrutura, uma vez que, na estrutura de repetição **for** a quantidade de laços fica mais evidente, e na estrutura da repetição **while** as repetições não são controladas por um mecanismo específico, permitindo o desenvolvimento mais flexível e criativo.

Com uma estrutura de repetição **while**, consegue criar um algoritmo que calcule a raiz quadrada, utilizando o Método de Newton para Cálculo de Raízes de Função:

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>Página JSP Estrutura de Repetição</title>
7  </head>
8  <body>
9      <%
10         double numero = 64.0;
11         double chuteado = 0.0;
12         // pode ser qualquer valor diferente da variável numero
13         double chute = 1.0;
14         while (chute != chuteado) {
15             chuteado = chute;
16             chute = (chute + numero / chute) / 2;
17         }
18         out.print("Raiz de "+numero+"<br>: "+chute);
19     %>
20 </body>
21 </html>

```

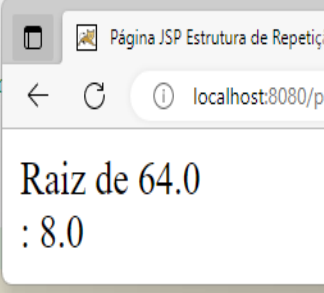


Relevante observar que a linguagem Java detém uma classe chamada Math que utilizou a estrutura de repetição em seu método sqrt, possibilitando o cálculo da raiz quadrada de forma simplificada, desta maneira o algoritmo abaixo terá o mesmo resultado que o algoritmo do exemplo recentemente abordado:

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>Página JSP Estrutura de Repetição</title>
7  </head>
8  <body>
9      <%
10         double numero = 64.0;
11         out.print("Raiz de "+numero+"<br>: "+Math.sqrt(numero));
12     %>
13 </body>
14 </html>

```



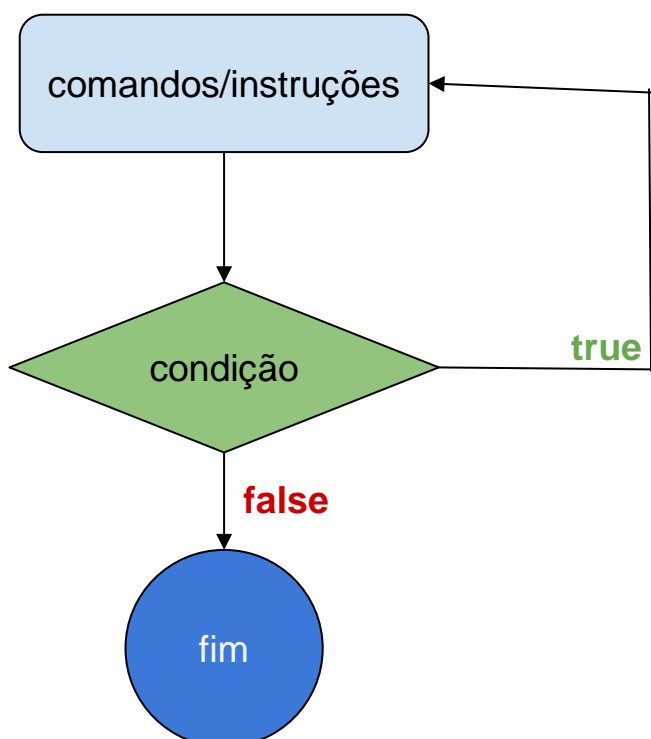
4. Estrutura de repetição - do while

A sintaxe da estrutura da repetição **do while** é a seguinte:

```
do {  
    instruções/comandos;  
}while (condição);
```

A diferença entre a estrutura de repetição **do while** e a estrutura de repetição **while** está no momento da execução da condição para execução das instruções/comandos, ou seja, na estrutura de repetição **do while** as instruções/comandos sempre serão executadas pelo menos uma vez.

Analise o diagrama da estrutura de repetição **do while** :



No exemplo abaixo nota-se que o resultado da condição da estrutura de repetição **do while** é **false**, mas ainda sim as instruções da estrutura são executadas uma vez.

The screenshot shows a code editor with the following JSP code:

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta http-equiv="Content-Type" content="text/html; charset="
6     <title>Página JSP Estrutura de Repetição</title>
7 </head>
8 <body>
9     <%
10         double contador = 0;
11         do{
12             out.print("Valor do contador:"+contador);
13             contador--;
14         }while(contador>0);
15     %>
16 </body>
17 </html>
    
```

Below the code editor, a browser window is shown with the title "Página JSP Estrutura de Repetição" and the URL "localhost:8080/primeiro_projeto". The browser displays the output: "Valor do contador:0.0".

5. Estrutura de dados -> coleção sequencial -> Array ou Vetor

Uma estrutura de dados que permita o armazenamento de uma coleção indexada de elementos na memória volátil de uma estrutura computacional é nominada na linguagem Java como **array**.

O array permite o armazenamento de uma coleção de dados, mas uma variável também, portanto a grande diferença é que um array permite o armazenamento indexado dos dados, o que na prática significa uma coleção de variáveis indexadas.

Vale listar algumas características dos arrays na linguagem Java:

- Os arrays são alocados dinamicamente;
- Uma variável array pode ser declarada como qualquer outra variável, mas com [] após o tipo de dados declarado;
- Arrays são objetos e podemos descobrir seu tamanho usando o atributo length do objeto;
- Os elementos em um array são indexados e o primeiro índice é sempre o zero;
- Um array pode ter uma ou mais dimensões, ou seja, unidimensional, bidimensional ou multidimensional;
- O tamanho de um array sempre deve ser um valor int, e nunca um long, short ou byte;

CURSO TÉCNICO EM INFORMÁTICA PARA INTERNET

- Podemos usar arrays como um campo estático, variável local ou ainda como parâmetros em métodos;
- Os tipos de array implementam as interfaces Cloneable e java.io.Serializable;
- Um array pode conter tipos de dados primitivos ou ainda objetos de uma classe

A sintaxe de declaração de uma array é a seguinte:

```
tipo nomeDoArray[ ];
```

```
tipo[ ] nomeDoArray;
```

O tipo determina qual será o tipo de dados de todos os elementos que serão armazenados nas posições do array, pois a estrutura é homogênea, ou seja, uma array do tipo `int` terá todos os elementos do tipo `int`, já um array do tipo `String` terá todos os elementos tipo `String`, como exemplifica-se:

```
int numeros[ ];
```

```
String palavras[ ];
```

Uma coisa é declarar o array, mesmo que criar uma referência, outra coisa é instanciar, pois quando uma instância é criada, pelo menos duas situações são definidas, a dimensão do array e o espaço na memória volátil da estrutura computacional.

É possível declarar e instanciar um array na mesma instrução, como exemplifica-se:

Sintaxe genérica:

```
tipo nomeDoArray[ ] = new tipo[dimensãoDoArray];
```

Sintaxe funcional:

```
int numeros[ ] = new int[4];
```

O exemplo da sintaxe funcional pode ser representado com o seguinte diagrama:

numeros	
índice	valor
0	
1	
2	
3	

Observa-se que o array `numeros` tem o comprimento 4, com índice começando com 0 e terminando com 3, possibilitando o armazenamento de 4 dados do tipo `int`.

Se a instrução `numeros[2] = 187;` for executada haverá a seguinte modificação no array:

numeros	
índice	valor
0	
1	
2	187
3	

Percebe-se que para armazenar um dado/valor em um array, basta utilizar o nome/referência do array e indicar a posição desejada dentro dos colchetes, conforme mais um exemplo:

`numeros[0] = 308;`

numeros	
índice	valor
0	308

1	
2	187
3	

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>Página JSP Estrutura de Repetição</title>
7  </head>
8  <body>
9  <%
10     int numeros[ ] = new int[4];
11     numeros[2] = 187;
12     numeros[0] = 308;
13     out.print("<br>Valor na posição 0 : "+numeros[0]);
14     out.print("<br>Valor na posição 1 : "+numeros[1]);
15     out.print("<br>Valor na posição 2 : "+numeros[2]);
16     out.print("<br>Valor na posição 3 : "+numeros[3]);
17
18 %>
19 </body>
20 </html>

```

Valor na posição 0 : 308
 Valor na posição 1 : 0
 Valor na posição 2 : 187
 Valor na posição 3 : 0

Outra forma de declarar e instanciar um array está ligada à situação em que se sabe quais são os valores iniciais do array, conhecida como array literal. A sintaxe para essa ocasião é a seguinte:

tipo nomeDoArray[] = {valor1, valor2, valor3, valorX};

Sintaxe funcional:

String palavras[] = {"bala", "chiclete", "pirulito"};

Diagrama resultante:

palavras	
índice	valor
0	bala

1	chiclete
2	pirulito

Importante entender que o array palavras detém o comprimento ou length 3, com índices de 0 até 2, sendo que para acessar os dados contidos no array basta utilizar o nome do array com colchetes ao final indicando a posição/índice do dado desejado, como exemplifica-se:

palavras[1] detém o dado **chiclete**

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>Página JSP Estrutura de Repetição</title>
7 </head>
8 <body>
9 <%
10     String palavras[ ] = {"bala", "chiclete", "pirulito"};
11     out.print("<br>Valor na posição 0 : "+palavras[0]);
12     out.print("<br>Valor na posição 1 : "+palavras[1]);
13     out.print("<br>Valor na posição 2 : "+palavras[2]);
14 %>
15 </body>
16 </html>

```



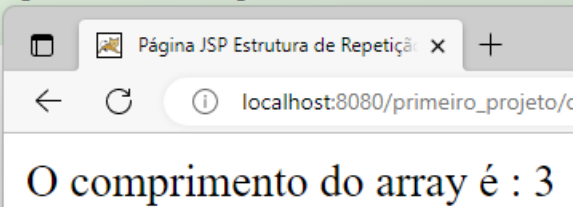
Como já foi mencionado, um array é um objeto, logo tem atributos e métodos. Um atributo importante para boa parte das operações com arrays é o length. O atributo length é responsável por conter a informação sobre o tamanho ou dimensão do array.

O array palavras utilizado no exemplo acima detém o comprimento 3, o que pode ser ratificado com o comando palavras.length, conforme exemplo abaixo:

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>Página JSP Estrutura de Repetição</title>
7  </head>
8  <body>
9  <%
10     String palavras[ ] = {"bala", "chiclete", "pirulito"};
11     out.print("O comprimento do array é : "+palavras.length);
12 %>
13 </body>
14 </html>
15
16

```



Por ora vale ainda salientar que para acessar o valor contido no último elemento de um array pode-se utilizar no espaço de indicação o índice a seguinte operação aritmética:

nomeDoArray[nomeDoArray.length - 1]

Exemplo funcional:

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>Página JSP Estrutura de Repetição</title>
7  </head>
8  <body>
9  <%
10     String palavras[ ] = {"bala", "chiclete", "pirulito"};
11     out.print("O último elemento contém : "+palavras[palavras.length-1]);
12 %>
13 </body>
14 </html>
15
16
17

```



Referências

- ★ <https://pet-comp-ufsc.github.io/tutorials/langs/java/for-while.html>
- ★ <http://www.inf.ufes.br/~vitorsouza/archive/2020/wp-content/uploads>
- ★ https://en.wikipedia.org/wiki/Newton%27s_method
- ★ <https://beginnersbook.com/2015/03/do-while-loop-in-java-with-example/>