Estruturas de repetição

> Laços de repetição, também conhecidos como laços de iteração ou simplesmente loops, são comandos que permitem iteração de código, ou seja, que comandos presentes no bloco sejam repetidos diversas vezes.
>
> https://diegomariano.com/lacos-de-repeticao-2/

Laços ou repetições são representados pelas seguintes estruturas:

- **For** (para);
- **While** (enquanto);
- **Do While** (faça enquanto).

For

O comando `for` (na tradução literal para a língua portuguesa “para”) permite que uma variável contadora, seja testada e incrementada a cada iteração, sendo essas informações definidas na chamada do comando. O comando **for** recebe como entrada uma variável contadora, a condição para continuar a execução e o valor de incrementação.

A estrutura de sintaxe do controle de repetição `for` é exibida abaixo:

```
//estrutura do controle de fluxo for

for (bloco de inicialização; expressão booleana de validação; bloco de atualização)
{
     // comando que será executado até que a
     // expressão de validação torne-se falsa
}
```

Vamos imaginar que Joãozinho precisa contar até 20 carneirinhos, para pegar no sono:

```
// ExemploFor.java
public class ExemploFor {
        public static void main(String[] args) {
                for(int carneirinhos = 1 ; carneirinhos <=20; carneirinhos ++) {
                        System.out.println(carneirinhos + " - Carneirinho(s)");
                }
        }
}
```

Vamos explicar a estrutura do código acima:

For position

1. int carneirinhos = 1 ; → O programa entende que a variável carneirinhos, começa com o valor igual a 1 e isso acontece somente uma vez;
2. carneirinhos < = 20; → O programa verifica se a variável carneirinhos, ainda é menor que 20;
3. O programa começa a executar o algoritmo, no nosso caso, imprimir a quantidade de carneirinhos em contagem;
4. carneirinhos ++ → O programa aumenta em mais 1, o valor da variável carneirinhos;
5. O fluxo é finalizado, quando a variável carneirinhos for igual a 20.

```
// Outras estruturas

//estrutura 1
for(int carneirinhos = 1 ; carneirinhos <=20; carneirinhos ++) {
     System.out.println(carneirinhos + " - Carneirinho(s)");
}

//estrutura 2
//o que importa é somente o bloco condicional
int carneirinhos = 1;
for( ; carneirinhos <=20; ) {
     System.out.println(carneirinhos + " - Carneirinho(s)");
     carneirinhos ++;
}

//for( somente 1x ; deve ser uma expresão boolean; acontecerá a cada final da execução ) { }
```

Também usamos o controle de fluxo **`for`** , para interagir sobre arrays e coleções:

```
// ExemploFor.java
public class ExemploFor {
public static void main(String[] args) {
```

```
        String alunos[] = { "FELIPE", "JONAS", "JULIA", "MARCOS" };

        for (int x=0; x<alunos.length; x++) {
                System.out.println("O aluno no indice x=" + x + " é " + alunos[x]);
        }

}
}
```

> Observe que, como os arrays começam com índice igual a **0 (zero)**, iniciamos a nossa variável `x` também com valor zero e validamos a quantidade de repetições, a partir da quantidade de elementos do array.

> ⓘ Fala a verdade: Depois desta explicação deu até sono hein!? 😴😴

For Each

O uso do **`for / each`** está fortemente relacionado, com um cenário onde contenha um array ou coleção, e assim , a interação é baseada nos elementos da coleção.

```
// ExemploFor.java
public class ExemploFor {
public static void main(String[] args) {
        String alunos [] =  {"FELIPE","JONAS","JULIA","MARCOS"};

        //Forma abreviada
        for(String aluno : alunos) {
          System.out.println(alunos);
        }

}
}
```

1. `String aluno : alunos` → De forma abreviada, é criada uma variável `nome` obtendo um elemento do vetor a cada ocorrência;
2. A impressão de cada aluno é realizada.

Break e Continue

Break significa quebrar, parar, frear, interromper. E é isso que se faz, quando o Java encontra esse comando pela frente. **Continue**, como o nome diz, ele 'continua' o laço. O comando **`break`** interrompe o laço, já o **`continue`** interrompe somente a iteração atual.

```
// class ExemploBreakContinue.java
public class ExemploBreakContinue {
        public static void main(String[] args) {

        for(int numero = 1; numero <=5; numero ++){
                if(numero==3)
                        break;

                System.out.println(numero);
```

```
        }
        //Qual a saída no console ?

    }
}


// class ExemploBreakContinue.java
public class ExemploBreakContinue {
        public static void main(String[] args) {

        for(int numero = 1; numero <=5; numero ++){
                if(numero==3)
                        continue;

                System.out.println(numero);

        }
        //Qual a saída no console ?

    }
}
```

> Observem que sempre o **break** e **continue** ``, está condicionado a um critério de negócio.

While

O laço **while** (na tradução literal para a língua portuguesa “enquanto”) determina que, enquanto uma condição for válida, o bloco de código será executado. O laço **while** , testa a condição antes de executar o código, logo, caso a condição seja inválida no primeiro teste o bloco nem será executado.

A estrutura de sintaxe, do controle de repetição **while** é exibida abaixo:

```
//estrutura do controle de fluxo while

while (expressão booleana de validação)
{
     // comando que será executado até que a
     // expressão de validação torne-se falsa
}
```

Joãozinho recebeu R$ 50,00 de mesada e foi em uma loja de doces gastar todo o seu dinheiro, logo, enquanto o valor dos doces não igualar a R$ 50,00 ele foi adicionando itens no carrinho.

```
// ExemploWhile.java
import java.util.concurrent.ThreadLocalRandom;
public class ExemploWhile {
public static void main(String[] args) {
        double mesada = 50.0;
        while(mesada>0) {
            Double valorDoce = valorAleatorio();
            if(valorDoce > mesada)
                valorDoce = mesada;

            System.out.println("Doce do valor: " + valorDoce + " Adicionado no carrinho");
            mesada = mesada - valorDoce;
        }
        System.out.println("Mesada:" + mesada);
        System.out.println("Joãozinho gastou toda a sua mesada em doces");

        /*
        * Não se preocupe quanto a formatação de valores
        * Iremos explorar os recursos de formatação em breve !!
        */
   }
   private static double valorAleatorio() {
        return ThreadLocalRandom.current().nextDouble(2, 8);
   }
}
```

Do While

O laço **do / while** (na tradução literal para a língua portuguesa “faça...enquanto”), assim como o laço while, considera que, enquanto uma determinada condição for válida, o bloco de código será executado. Entretanto, **do / while** testa a condição após executar o código, sendo assim, mesmo que a condição seja considerada inválida, no primeiro teste o bloco será executado pelo menos uma vez.

A estrutura de sintaxe do controle de repetição **do / while** é exibida abaixo:

```
//estrutura do controle de fluxo do while

do
{
    // comando que será executado até que a
     // expressão de validação torne-se falsa
}
while (expressão booleana de validação);
```

Joãozinho resolveu ligar para o seu amigo, dizendo que hoje se entupiu de comer docinhos:

```
// ExemploDoWhile.java

import java.util.Random;
public class ExemploDoWhile {
public static void main(String[] args) {
        public static void main(String[] args) {
                System.out.println("Discando...");

                do {
                        //excutando repetidas vezes até alguém atender
                        System.out.println("Telefone tocando");

                }while(tocando());

                System.out.println("Alô !!!");
        }
        private static boolean tocando() {
                boolean atendeu = new Random().nextInt(3)==1;
                System.out.println("Atendeu? " + atendeu);
                //negando o ato de continuar tocando
                return ! atendeu;
        }
}
```

Controle de fluxo - Previous
Estruturas condicionais

Next - Controle de fluxo
Estruturas excepcionais

Last modified 1mo ago