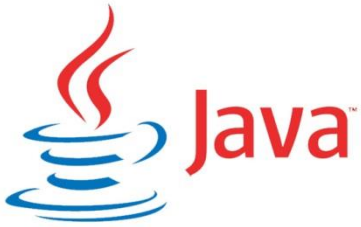
Lógica de **P**rogramação

Unidade 13 – Laços de Repetição (while)



QI ESCOLAS E FACULDADES
Curso Técnico em Informática

SUMÁRIO

INTRODUÇÃO

Os laços permitem que um determinado bloco de comandos seja executado repetidamente a partir de uma condição.

Este tipo de instrução é utilizada em menus (onde o programa irá repetir enquanto o usuário quiser utilizá-lo), em pesquisas (onde podemos determinar a quantidade de pessoas que irão interagir com o programa respondendo a enquete) e, ou até mesmo em métodos que precisam ser executados várias vezes até encontrar a resposta desejada, por exemplo: fatorial de um número. Estas estruturas também ajudam a evitar que se escreva o mesmo comando várias vezes.

> *Laço de repetição também é conhecido por loop, estruturas de repetição ou simplesmente laços.*

TIPOS DE LAÇOS DE REPETIÇÃO

Cada linguagem de programação oferece algumas estruturas para desenvolver algoritmos com laços de repetição. Em Java, trabalhamos basicamente com os comandos:

while → enquanto

do while → faça enquanto

for → para

EXECUÇÃO DE UM LAÇO DE REPETIÇÃO

Temos dois tipos de execução: execução onde temos um **laço determinado** ou execução com um **laço indeterminado**.

Laços Determinados

Os laços determinados são aqueles nos quais nós como programadores temos o controle de quantas vezes o *loop* será executado, ou seja, sabemos o número de vezes que a instrução irá repetir, temos o controle do início e do fim do laço.

Exemplos de Laços Determinados:

- Uma enquete onde o objetivo é entrevistar 50 pessoas (por exemplo).

Neste laço sabemos que a enquete será repetida 50 vezes, e que o início será na 1ª pessoa entrevistada, e o fim será na 50ª pessoa entrevistada.

- Um método que retorne a tabuada de um número

Sabemos que as instruções deste método serão repetidas 10 vezes, pois a tabuada de um número é o número multiplicado por um, dois, três, quatro... até dez. O início será dado no número 1 e o término no número 10.

Laços Indeterminados

Os laços indeterminados são aqueles nos quais não temos controle de quantas vezes serão executados, por tanto sabemos o seu início, porém não sabemos o seu fim.

Exemplos de Laços Indeterminados:

- O usuário escolhe se deseja sair ou deseja testar o programa mais vezes.

Até agora nossos programas executam apenas uma vez, para testarmos novamente temos que fechar o terminal e reabri-lo. Para não termos esse trabalho, podemos apenas colocar um *loop* no programa e fazer com que antes de terminar o programa o mesmo mostre na tela a seguinte mensagem:

```
"Deseja continuar?
Digite 1 para continuar ou 0 para sair: →"
```

Nesta situação sabemos o início do programa, porém não sabemos quando irá terminar, pois o usuário poderá testar várias vezes.

- Programas com menus

Outra situação em que temos laços indeterminados é em programas com menu, nestes programas sabemos que o início é quando o usuário abrir o mesmo, porém não sabemos quantas vezes ele irá utilizar o menu, o que ele irá fazer. Se irá cadastrar um dado, visualizar um dado, excluir, ou simplesmente sair do sistema.

INSTRUÇÃO WHILE (ENQUANTO)

O laço de repetição ***while*** caracteriza-se por ter seu teste de execução antes de iniciar o *loop*. Neste tipo de laço nem sempre temos a execução dos comandos, ou seja, nem sempre ele entra no *loop*.

Sintaxe da estrutura while

```
while(condição){
	<instruções>;
}
```

Condição – O *while* executará as instruções enquanto a condição nos parênteses for verdadeira! No momento que a condição for falsa ele para de executar as instruções dentro do bloco *while*.

Instruções – Dentro de um *while* podemos utilizar qualquer tipo de instrução, ou seja, podemos: mostrar uma mensagem, declarar variáveis, utilizar *if*(){}, utilizar outro *while*(){}, e assim por diante.

Exemplo da instrução while

Imagine sua rotina da semana, durante a semana você acorda, toma café, pega o ônibus e chega ao trabalho. Se fêssemos pensar em um *loop*, como seria?

Português:

```
enquanto(não chegar no fim de semana)faça{
	Acordar
	tomar café
	Pegar o ônibus
	trabalhar
	Voltar para casa
}
```

Sintaxe no programa:

```
Pessoa p1 = new Pessoa();
int dia = 2; //2 significa segunda-feira
while(dia<7){ //7 significa sábado
	p1.acordar();
	p1.tomarCafe();
	p1.pegarOnibus();
	p1.trabalhar();
	p1.voltarParaCasa();
	dia = dia+1; //isso significa que no final de cada loop, irá aumentar um dia na semana
}
```

```
System.out.println("Finalmente fim de semana!");
```

Analisando o exemplo acima, podemos perceber que iniciamos a semana na segunda, onde definimos dia = 2. O *while* irá testar se o dia é menor que 7, ou seja, enquanto o dia for menor que 7 (sábado) iremos executar a rotina da semana.

Neste programa o código irá executar 5 vezes, ele começará na segunda (2), passará para dia 3 (terça), dia 4 (quarta), dia 5 (quinta) e dia 6 (sexta), quando chegar no dia 7 irá sair do *loop* e retornar na tela: **"Finalmente fim de semana!".** O que garante que ele aumentará de 1 em 1 a cada volta é o comando "dia=dia+1".

Vamos considerar o exemplo de uma classe Numero, tendo como atributo um valor privado, seus métodos set/get e um método para ver o fatorial do número. Lembre-se que o fatorial é a multiplicação do número por todos os seus antecessores. Exemplo: **Fatorial do número 4 é 24**, pois é 1*2*3*4 = 24.

UML da classe Numero:

Numero
-valor: int
+getValor():int +setValor(valor:int):void +calcularFatorial ():int

```
public class Numero{
    private int valor;

    public void setValor(int valor){
        this.valor = valor;
    }
    public int getValor(){
        return this.valor;
    }
    public int calcularFatorial(){
        int cont = 1;
        int fatorial = 1;
        while(cont<=this.valor){
            fatorial = fatorial * cont;
            cont = cont + 1;
        }
        return fatorial;
    }
}
```

TESTE DE MESA

Um teste de mesa serve para analisarmos o código e testar o funcionamento de nosso programa, o fluxo de informação.

Analisando o método e fazendo um teste de mesa

```
1. public int calcularFatorial (){
2.     int cont = 1;
3.     int fatorial = 1;
4.
5.     while(cont<=this.valor){
6.           fatorial = fatorial * cont;
7.           cont = cont + 1;
8.     }
8.
10.    return fatorial;
11.}
```

Com o teste de mesa podemos observar o que acontecerá na execução do método, note que no **exemplo** do teste de mesa utilizaremos o valor **4** (imagine que este valor o usuário digitou), observe a tabela abaixo onde colocamos: **this.valor, cont e fatorial** como título, correspondendo às variáveis que foram alocadas na memória.

this.valor	cont	fatorial
4	1	1

O **atributo valor** (this.valor) está sendo utilizado para guardar o valor que o usuário irá informar, neste exemplo o número **4**.

A **variável cont** será uma variável auxiliar para termos controle de quantas vezes o código será executado, neste caso, ela inicializa em **1** e irá até o valor. Ou seja, em nosso exemplo será: 1, 2, 3 e 4, quando ela chegar em 5 não irá executar mais as instruções do laço.

A **variável fatorial** será uma variável auxiliar para acumular os valores parciais do cálculo, o resultado da multiplicação do valor com seus antecessores. Perceba que ela inicializa em **1** pois este valor será multiplicado pelo valor do contador.

Perceba que cada linha da tabela guarda um valor que as variáveis vão recebendo ao longo do loop e os valores riscados são valores não estão mais na memória, já foram substituídos pelos valores abaixo.

Na **primeira** execução do *loop*, a condição dará verdadeira, pois o *cont* é menor ou igual ao *this.valor* (1<=4). Portanto, ele executará na sequência os comandos:

```
fatorial = fatorial * cont; (fatorial = 1 * 1)
cont = cont + 1; (cont = 1 + 1)
```

Após isto, nossa tabela ficará da seguinte forma:

valor	cont	fatorial
4	~~1~~	~~1~~
	2	1

- Observe que os valores anteriores são substituídos, valendo o último valor que a variável recebeu.

Ao chegar na chave (linha 8) ele retorna ao *while* para testar novamente a condição (linha 5). Logo temos: *cont<=this.valor* **=>** 2<=4 **=>** verdadeiro, portanto executará novamente os comandos:

```
fatorial = fatorial * cont; (fatorial = 1 * 2)
cont = cont + 1; (cont = 2 + 1)
```

Após isto, nossa tabela ficará da seguinte forma:

valor	cont	fatorial
4	~~1~~	~~1~~
	~~2~~	~~1~~
	3	2

Retornando à linha 5 e testando novamente a condição temos: *cont<=this.valor* **=>** 3<=4 **=>** verdadeiro, portanto executará novamente os dois comandos:

```
fatorial = fatorial * cont; (fatorial = 2 * 3)
cont = cont + 1; (cont = 3 + 1)
```

Após isto, nossa tabela ficará da seguinte forma:

valor	cont	fatorial
4	~~1~~	~~1~~
	~~2~~	~~1~~
	~~3~~	~~2~~
	4	6

Retornando à linha 5 e testando novamente a condição temos: *cont<=this.valor* **=>** 4<=4 **=>** verdadeiro, portanto executará novamente os dois comandos:

```
fatorial = fatorial * cont; (fatorial = 6 * 4)
cont = cont + 1; (cont = 4 + 1)
```

Após isto, nossa tabela ficará da seguinte forma:

valor	cont	fatorial
4	~~1~~	~~1~~
	~~2~~	~~1~~
	~~3~~	~~2~~
	~~4~~	~~6~~
	5	**24**

Retornando à linha 5 e testando novamente a condição temos: *cont<=this.valor* **=>** 5<=4 **=>** FALSO, portanto aqui encerra a execução do *loop* e o método retorna o último valor armazenado na variável fatorial que é **24**.