Aula VII - Desenvolvimento de Aplicativo I

Lógica
Estruturas de Repetição
Laços de Repetição
While - Enquanto
Do - Faça
Do while - Faça enquanto
For - Para

Fontes
https://tableless.com.br/java-estruturas-de-repeticao/
https://pt.wikibooks.org/wiki/Java/Comandos_de_itera%C3%A7%C3%A3o
http://ead.qi.edu.br/pluginfile.php/11865/mod_resource/content/0/Unidade%2015%20-%20La%C3%A7os%20de%20Repeti%C3%A7%C3%A3o%20-%20FOR.pdf
http://distancia.qi.edu.br/mod/book/view.php?id=13695&chapterid=9245 - Disciplina de Linguagem de Programação I - Professor Eduardo Reus
http://ead.qi.edu.br/pluginfile.php/11865/mod_resource/content/0/Unidade%2015%20-%20La%C3%A7os%20de%20Repeti%C3%A7%C3%A3o%20-%20FOR.pdf
http://excript.com/java/operador-incremento-decremento-java.html
http://ead.qi.edu.br/pluginfile.php/11864/mod_resource/content/0/Unidade%2014%20-%20La%C3%A7os%20de%20Repeti%C3%A7%C3%A3o%20-%20DO%20WHILE.pdf

---

Laços de Repetição

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

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

Tipos de execução:

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. 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.

Os laços indeterminados: são aqueles nos quais **não temos controle de quantas vezes serão executados, portanto 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 foram executados 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: →"
```

Resumindo!

As estruturas de repetição também são conhecidas como laços (loops) e são utilizadas para executar, repetidamente, uma instrução ou bloco de instrução enquanto determinada condição estiver sendo satisfeita.
Qualquer que seja a estrutura de repetição, ela contém quatro elementos fundamentais: inicialização, condição, corpo e iteração.
A inicialização compõe-se de todo código que determina a condição inicial da repetição.
A condição é uma expressão booleana avaliada após cada leitura do corpo e determina se uma nova leitura deve ser feita ou se a estrutura de repetição deve ser encerrada.
O corpo compõe-se de todas as instruções que são executadas repetidamente.
A iteração é a instrução que deve ser executada depois do corpo e antes de uma nova repetição.

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, traduzindo, o **While é utilizado para construir uma estrutura de**

repetição que executa, repetidamente, uma única instrução ou um bloco delas "enquanto" uma expressão booleana for verdadeira.

Sintaxe básica:

enquanto a condição for verdadeira {
executa as instruções
}

Exemplos:

```
int x = 0;
while (x < 10) {
   System.out.println("Item " + x);
   x++;
}
```

```
public class Lacos {
    public static void main(String[] args) {
        int x = 15;
        while (x < 18) {
            System.out.println("Você não tem permissão para entrar");
            x ++;
        }
    }
}
```

Exemplos:

```
public class UsoWhile {
    public UsoWhile() {
        int i=0;
        while (++i < 50) {
            System.out.println(i);
        }
    }
}
```

Podemos realizar o incremento do contador na comparação do While, como nos exemplos da tabela.
Os operadores de incremento e decremento são operadores compostos por o **símbolo de adição seguido de outro símbolo de adição, ou então, o símbolo de subtração seguido por outro símbolo de subtração**.

```java
public class UsoWhile2 {
    public UsoWhile2() {
        int i=(-1);
        while (i++ < 50) {
            System.out.println(i);
        }
    }
}
```

```java
public class UsoWhile3 {
    public UsoWhile3() {
        int x = 0;
       while (x < 10) {
    System.out.println("Item " + x);
    x++;
        }
    }
}
```

Exemplo de código com Incremento:

```java
public class Incremento {
    public static void main(String[] args) {
        //Operadores de Incremento
        int i = 0;
        int y = 0;
        i += 1;//++i PRE-incremento
        /*PRE - o valor sera incrementado antes da instr
        onde a nossa expressão estiver contida
        POS - o valor será incrementado após a execução
        onde a nossa expressão estiver contida */
        int a = 0;
        int b = 0;
        System.out.println(" - - - - - - - -");
        System.out.println( --a );//PRE-DECREMENTO
        System.out.println( a-- );//POS-DECREMENTO
    }
}
```

Exemplo 2:

```java
public class Fatorial {
    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;
    }
```

Do While (faça enquanto):

A estrutura de repetição **do-while é uma variação da estrutura while**. Existe uma diferença sutil, porém importante, entre elas. **Em um laço while, a condição é testada antes da primeira execução das instruções que compõem seu corpo. Desse modo, se a condição for falsa na primeira vez em que for avaliada, as instruções desse laço não serão executadas nenhuma vez. Em um laço do-while, por outro lado, a condição somente é avaliada depois que suas instruções são executadas pela primeira vez, assim, mesmo que a condição desse laço seja falsa antes de ele iniciar, suas instruções serão executadas pelo menos uma vez.**

Condição: O do 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 do while.

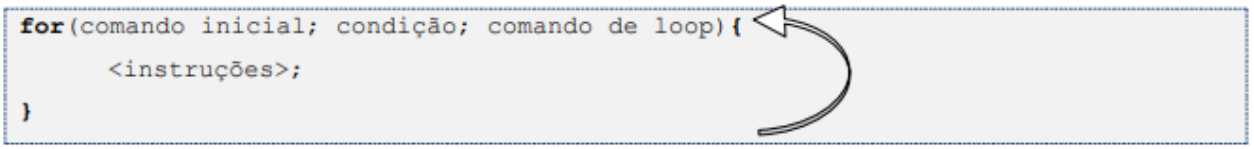
Observe que na sua primeira execução ele não analisa nenhuma condição: executa primeiro e depois analisa. Instruções – Dentro de um do while podemos utilizar qualquer tipo de instrução, ou seja, podemos: mostrar uma mensagem, declarar variáveis, utilizar if(){}, utilizar um while(){} e assim por diante.

```
public class TesteIdade {
    public static void main(String[] args){
        int x = 15;
        do{
            System.out.println("Você não tem permissão para entrar");
            x++;
        }while (x < 18);
        }
    }
```

For (para):

O for (**para, em inglês - quer dizer: "para este caso... fazer..."**) pode conter apenas uma instrução no seu corpo. Neste caso não é necessário abrir um bloco. Isso é assim porque o "for" já implementa alguns comandos na sua assinatura, ou seja, no seu cabeçalho, como a inicialização da variável e o passo da re
petição, ou seja, o incremento/decremento da variável (executado sempre no fim de cada ciclo). **O laço for é uma estrutura de repetição compacta.** Seus elementos de **inicialização, condição e iteração são reunidos na forma de um cabeçalho e o corpo é disposto em seguida.**

O **laço for e o laço while são apenas formas diferentes de uma mesma estrutura básica de repetição.** Qualquer laço for pode ser transcrito em termos de um laço while e vice-versa. Do mesmo modo que em um laço while, se a condição de um laço for já é falsa logo na primeira avaliação que se fizer dela, as instruções contidas em seu corpo jamais serão executadas.



Vamos analisar a sintaxe da **estrutura for**.

Observe que na primeira linha, dentro dos parênteses, ao invés de termos apenas a condição como acontecia com o while e do while, **temos agora três comandos separados por ponto e vírgula.**

```
public class ComandoPara {
    public static void main(String[] args){
        for(int x = 0; x < 100; x ++){
        System.out.println("Valor: " + x);
    }
    }
}
```

Comando inicial: Este comando executa apenas uma vez, quando o código entra no loop. Aqui podemos declarar uma variável, por exemplo, como é muito comum.
Condição: O mesmo tipo de condição que montamos em while e do while. Se ela for verdadeira, as instruções serão executadas. Se for falsa, o loop encerra.
Comando de loop: Este comando é executado sempre que o **for** completa uma volta, ou seja, na primeira execução ele pula este comando.

Exemplo2:

```
public class Numero {
    private int valor;
    //construtor
    public Numero (int valor){
        this.valor= valor;
    }
    //metodos
    public int getValor(){
        return valor;
    }
     public void setValor(int valor){
     this.valor = valor;
    }
    //método para calcular
    public int calcularFatorial() {
        //criando uma var x
        int x;
        //estou fazendo a minha resposta iniciando em 1
        int resposta = 1;
        //começando laço de repetição "for", primeiro o valor inicial (neste cas
        //depois a condição que quer dizer até
        //e a cada volta (da execução) X++
        //equivalente a fazer X + 1
        for (x = 1; x <= valor; x ++) {
        //resposta recebendo ela mesmo * pela variável X
        resposta = resposta * x;
    }
        //retorna a resposta
        return resposta;
    }
}
```

```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int valor;

        System.out.print("Informe um valor: ");
        valor = teclado.nextInt();

        Numero n = new Numero(valor);

        System.out.println("Fatorial: " + n.calcularFatorial())

    }
}
```