ESTRUTURA DE CONDIÇÃO E REPETIÇÃO

As estruturas condicionais possibilitam ao programa tomar decisões e alterar o seu fluxo de execução. Isso possibilita ao desenvolvedor o poder de controlar quais são as tarefas e trechos de código executados de acordo com diferentes situações, como os valores de variáveis.

As estruturas condicionais geralmente analisam expressões booleanas e, caso estas expressões sejam verdadeiras, um trecho do código é executado. No caso contrário, outro trecho do código é executado.

O if e o else

O if/else é uma estrutura de condição em que uma expressão booleana é analisada. Quando a condição que estiver dentro do if for verdadeira, ela é executada. Já o else é utilizado para definir o que é executado quando a condição analisada pelo if for falsa. Caso o if seja verdadeiro e, consequentemente executado, o else não é executado.

O if pode ser utilizado em conjunto com o else ou até mesmo sozinho, caso necessário.

No Java, a sintaxe do if é a seguinte:

```
if (condicaoBooleana) {
   codigo;
}
```

Uma **condição booleana** é qualquer expressão que retorne true ou false. Para isso, você pode usar os operadores <, >, <=, >= e outros.

Um exemplo:

```
int idade = 15;
if (idade < 18) {
   System.out.println("Não pode entrar");
}
```

Além disso, você pode usar a cláusula eles para indicar o comportamento que deve ser executado no caso da expressão booleana ser falsa:

```
int idade = 15;
if (idade < 18) {
   System.out.println("Não pode entrar");
} else {
   System.out.println("Pode entrar");
}
```

Você pode concatenar expressões booleanas por meio dos operadores lógicos **"E"** e **"OU"**. O **"E"** é representado pelo &&, e o **"OU"** é representado pelo ||.
Um exemplo seria verificar se ele tem menos de 18 anos **e** se não é amigo do dono:

```
int idade = 15;
```

```
boolean amigoDoDono = true;
if (idade < 18  &&  amigoDoDono == false) {
   System.out.println("Não pode entrar");
}
else {
   System.out.println("Pode entrar");
}
```

Esse código poderia ficar ainda mais legível, utilizando-se do operador de negação, o !. Esse operador transforma o resultado de uma expressão booleana de false em true, e vice-versa.

```
int idade = 15;
boolean amigoDoDono = true;
if (idade < 18  &&  !amigoDoDono) {
   System.out.println("Não pode entrar");
}
else {
   System.out.println("Pode entrar");
}
```

Repare na linha 3 que o trecho amigoDoDono == false virou !amigoDoDono.

Eles têm o mesmo valor.

Para comparar se uma variável tem o **mesmo valor** que outra variável ou que um valor, utilizamos o operador ==. Repare que utilizar o operador = dentro de um if retornará um erro de compilação, já que o operador = é o de atribuição.

```
int mes = 1;
if (mes == 1) {
        System.out.println("Você deveria estar de férias");
}
```

Operador if-then-else ternário

expressão ? declaração1 : declaração2

Onde expressão é qualquer expressão que o resultado seja um valor boolean. Se o resultado for true, a declaração 1 é executada e se for false a declaração 2 é executada.

Por exemplo:

```
resultado = valor==0 ? 0 : num/valor;
```

Na expressão acima é avaliado se valor é igual a zero. Caso positivo ratio recebe zero, senão resultado recebe num/valor.

O While

O while é um comando usado para fazer um **laço (loop)**, isto é, repetir um trecho de código algumas vezes. A ideia é que esse trecho de código seja repetido enquanto uma determinada condição permanecer verdadeira.

```
int idade = 15;
while (idade < 18) {
   System.out.println(idade);
```

```
    idade = idade + 1;
}
```
O trecho dentro do bloco do while será executado até o momento em que a condição idade < 18 passe a ser falsa.
E isso ocorrerá exatamente no instante em que idade == 18, o que não o fará imprimir 18.

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i = i + 1;
}
```
Já o while acima imprime de 0 a 9.

O For

Outro comando de **loop** extremamente utilizado é o for. A ideia é a mesma do while: fazer um trecho de código ser repetido, enquanto uma condição continuar verdadeira. Mas, além disso, o for isola também um espaço para inicialização de variáveis e o modificador dessas variáveis. Isso faz com que as variáveis relacionadas ao loop fiquem mais legíveis:

```
for (inicializacao; condicao; incremento) {
    codigo;
}
```

Um exemplo é:
```
for (int i = 0; i < 10; i = i + 1) {
    System.out.println("olá!");
}
```
Repare que esse for poderia ser trocado por:
```
int i = 0;
while (i < 10) {
    System.out.println("olá!");
    i = i + 1;
}
```

Porém, o código do for indica claramente que a variável i serve, em especial, para controlar a quantidade de laços executados. Quando usar o for? Quando usar o while? Depende do gosto e da ocasião.

Pós incremento ++

i = i + 1 pode realmente ser substituído por i++ quando isolado.

Porém, em alguns casos, temos essa instrução envolvida em, por exemplo, uma atribuição:
```
int i = 5;
int x = i++;
```

Qual é o valor de x? O de i, após essa linha, é 6.

O operador ++, quando vem após a variável, retorna o valor antigo e o incrementa (pós-incremento), fazendo x valer 5.
Se você tivesse usado o ++ antes da variável (pré-incremento), o resultado seria 6:

```
int i = 5;
int x = ++i; // aqui x valera 6.
```

Controlando loops

Apesar de termos condições booleanas nos nossos laços, em algum momento, podemos decidir parar o loop por algum motivo especial sem que o resto do laço seja executado.

```
for (int i = x; i < y; i++) {
   if (i % 19 == 0) {
      System.out.println("Achei um número divisível por 19 entre x e y");
      break;
   }
}
```

O código acima percorrerá os números de x a y e irá parar quando encontrar um número divisível por 19, uma vez que foi utilizada a palavra-chave break.

Da mesma maneira, é possível obrigar o loop a executar o próximo laço. Para isso, usamos a palavra-chave continue

```
for (int i = 0; i < 100; i++) {
   if (i > 50 && i < 60) {
      continue;
   }
   System.out.println(i);
}
```

O código acima não imprimirá alguns números. (Quais exatamente?)

Uma variável em Java tem o seu escopo de vida.

O escopo é a vida de uma variável em **Java**, tratando-se dos locais nos quais ela pode ser acessada. Em **Java**, o **escopo** de variáveis vai de acordo com o bloco onde ela foi declarada. A variável é criada no primeiro acesso a ela e destruída após o interpretador sair do bloco de execução ao qual ela pertence.

O escopo de vida de uma variável, é aquele lugar em que ela pode ser referenciada (chamada, manipulada), esse escopo, por sinal, é o mesmo em que ela é declarada. Isso significa que, fora desse escopo, a variável não pode ser referenciada, é como se ela não existisse. Por exemplo: Uma variável declarada dentro de um if, só é válida dentro do if. Uma variável declarada na inicialização do comando for, só é válida dentro dele.

Variáveis que são recebidas como parâmetros em métodos, também só são válidas dentro do método. Variáveis com o mesmo nome podem existir, desde que não estejam no mesmo escopo.

No Java, podemos declarar variáveis a qualquer momento. Porém, dependendo de onde você as declarou, ela valerá de um determinado ponto a outro.

```
// aqui, a variável i não existe.
int i = 5;
```

```
// a partir daqui, ela existe.
```

O **escopo da variável** é o nome dado ao trecho de código em que aquela variável existe e o lugar onde é possível acessá-la. Quando abrimos um novo bloco com as chaves, as variáveis declaradas ali dentro **só valem até o fim daquele bloco**.

```
// aqui, a variável i não existe.
int i = 5;
// a partir daqui, ela existe.
while (condicao) {
    // o i ainda vale aqui.
    int j = 7;
    // o j passa a existir.
}
// aqui, o j não existe mais, porém o i continua dentro do escopo.
```

No bloco acima, a variável j para de existir quando termina o bloco no qual ela foi declarada. Se você tentar acessar uma variável fora do escopo dela, ocorrerá um erro de compilação

O mesmo vale para um if:

```
if (algumBooleano) {
    int i = 5;
}
else {
    int i = 10;
}
System.out.println(i); // cuidado!
```

Aqui a variável i não existe fora do if e do else! Se você declarar a variável antes do if, haverá outro erro de compilação: dentro do if e do else, a variável está sendo redeclarada. Então, o código para compilar e fazer sentido fica:

```
int i;
if (algumBooleano) {
    i = 5;
}
else {
    i = 10;
}
System.out.println(i);
```

Uma situação parecida pode ocorrer com o for:

```
for (int i = 0; i < 10; i++) {
    System.out.println("olá!");
}
System.out.println(i); // cuidado!
```

Nesse for, a variável i morre ao seu término, não podendo ser acessada de fora do for e gerando um erro de compilação. Se você realmente quer acessar o contador depois do loop terminar, precisa de algo como:

```
int i;
for (i = 0; i < 10; i++) {
   System.out.println("olá!");
}
System.out.println(i);
```

Um bloco dentro do outro

Um bloco também pode ser declarado dentro de outro. Isto é, um if dentro de um for, ou um for dentro de um for, algo como:

```
while (condicao) {
   for (int i = 0; i < 10; i++) {
      // código
   }
}
```

O switch

A declaração switch é usada quando existem procedimentos diferenciados para determinados valores de uma única variável. Ela só pode ser usada para váriáveis que sejam de tipo primitivos de dados, mais especificamente para variáveis do tipo int (inteiros) e char (caracteres). Por exemplo:

```
int mes  = 4;
switch (mes) {
 case 1: System.out.println ("Janeiro");
                break;
case 2 : System.out.println ("Fevereiro");
                break;
case 3 : System.out.println ("Marco");
                break;
case 4: System.out.println ("Abril");
                break;
case 5 : System.out.println ("Maio");
                break;
case 6 : System.out.println ("Junho");
                break;
case 7: System.out.println ("Julho");
                break;
case 8 : System.out.println ("Agosto");
                break;
case 9 : System.out.println ("Setembro");
                break;
case 10 : System.out.println ("Outubro");
                break;
case 11 : System.out.println ("Novembro");
                break;
case 12 : System.out.println ("Dezembro");
                 break;
default : System.out.println ("Não existe mês correspondente!");
}
```

No exemplo acima, será impresso “Abril” como resultado na tela. Isto porque o valor da expressão é comparado com cada um dos valores das declarações case. Se uma correspondência for encontrada, a seqüência de código que vem depois da declaração case é executada. Se nenhum dos cases corresponder ao valor da expressão, então a declaração default é executada. Entretanto, a declaração default é opcional. O break faz com que a execução pule para o final do switch. Se não fosse colocado o break, a execução seria passada para o próximo case até encontrar um break.

Exemplos de códigos com variáveis

```java
public class Tipos_de_Dados {

    public static void main(String[] args) {
        System.out.println("Tipos de dados em Java: \n" +
                "\nMenor Byte: " + Byte.MIN_VALUE +
                "\nMaior Byte: " + Byte.MAX_VALUE +
                "\nMenor Short Int: " + Short.MIN_VALUE +
                "\nMaior Short Int: " + Short.MAX_VALUE +
                "\nMenor Int: " + Integer.MIN_VALUE +
                "\nMaior Int: " + Integer.MAX_VALUE +
                "\nMenor Long: " + Long.MIN_VALUE +
                "\nMaior Long:" + Long.MAX_VALUE +
                "\nMenor Float: " + Float.MIN_VALUE +
                "\nMaior Float: " + Float.MAX_VALUE +
                "\nMenor Double: " + Double.MIN_VALUE +
                "\nMaior Double: " + Double.MAX_VALUE);

    }

}
```

```java
public class Tipos_Primitivos {
    public static void main(String[] args) {
          byte tipoByte = 127;
          short tipoShort = 32767;
          char tipoChar = 'C';
          float tipoFloat = 2.6f;
          double tipoDouble = 3.59;
          int tipoInt = 2147483647;
          long tipoLong = 92233720368547758O7L;
          boolean tipoBooleano = true;
          System.out.println("Valor do tipoByte = " + tipoByte);
          System.out.println("Valor do tipoShort = " + tipoShort);
          System.out.println("Valor do tipoChar = " + tipoChar);
          System.out.println("Valor do tipoFloat = " + tipoFloat);
          System.out.println("Valor do tipoDouble = " + tipoDouble);
          System.out.println("Valor do tipoInt = " + tipoInt);
          System.out.println("Valor do tipoLong = " + tipoLong);
          System.out.println("Valor do tipoBooleano = " + tipoBooleano);
    }
}
```

Exercícios: fixação de sintaxe

1.Imprima todos os números de 150 a 300.

2.Imprima a soma de 1 até 1000.

3.Imprima todos os múltiplos de 3, entre 1 e 100.

4.Imprima os fatoriais de 1 a 10.

5.No código do exercício anterior, aumente a quantidade de números que terão os fatoriais impressos até 20, 30 e 40.

6.(Opcional) Escreva um programa em que, dada uma variável x com algum valor inteiro, temos um novo x de acordo com a seguinte regra:

•Se x é par, x = x / 2;

•Se x é impar, x = 3 * x + 1;

•Imprime x;

7.Escreva uma classe Java que contenha apenas o método main( ). Você deve passar como parâmetro para o método main( ) as coordenadas x, y de pontos. O método calcula a distância Euclidiana ente estes dois pontos e exibe o resultado na tela.

$$d(P,Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$