Curso de Java

aula 03

Prof. Anderson Henrique

TECH
soluções em ti

Estruturas de Controle

Qualquer linguagem de programação dispõe de meios para tornar possível a tomada de **decisões** ou a **repetição** de trechos/linhas de códigos.

Devemos entender por tomada de decisões a capacidade que o programa tem de efetuar um teste e executar uma ou outra tarefa de acordo com o resultado obtido.

Já a repetição de códigos diz respeito à capacidade de uma ou mais linhas do programa serem executadas um determinado número de vezes (loop).

De qualquer forma, esses dois recursos constituem a essência de um programa, pois tornam viável a construção de aplicativos com um alto grau de inteligência, no sentido de que sejam capazes de se ajustar às ações do usuário.

Assim como em C/C++, na linguagem Java esses comandos de decisão e de repetição são muito poderosos e ricos em flexibilidade.

Comando if

Essencialmente, ele efetua um teste comparativo utilizando uma expressão lógica e, dependendo do resultado (true/false), executa uma determinada ação. Essa ação pode ser constituída por uma ou mais linhas de instruções.

Sintaxe básica:

```
if(<expressão>){
        <instrução>;
}
```

Em **<expressão>** teremos uma expressão de teste válida. Na realidade, podemos ter várias expressões de teste conectadas por operadores lógicos. Já **<instrução>** representa a linha de comando que deve ser executada caso **<expressão>** resulte em um valor “true”.

Exemplo:

```
if(intIdade >= 18){
        System.out.println(“Tem maioridade”);
}
```

Comando else

Um comando que faz par com o **if**. Com ele podemos forçar o programa a executar uma tarefa alternativa caso o teste resulte em um valor “false”, ou seja, é uma segunda opção de decisão.

O comando **if** pode vir sem o **else**, mas o contrário não pode ocorrer.

Sintaxe básica:

```
if(<expressão>){
        <instrução1>;
else{
        <instrução2>;
}
```

Exemplo:

```
if(intIdade >= 18){
        System.out.println(“Tem maioridade”);
}else{
        System.out.println(“Não tem maioridade”);
}
```

No exemplo acima, estamos analisando se a idade de uma pessoa é maior ou igual a 18, se o resultado for “true”, o programa deverá executar a instrução que se encontra entre abre chaves { e o fecha chaves }, senão deverá executar a outra instrução correspondente.

Comando else if

Imagine a seguinte situação, você deverá construir um programa que analise a média escolar de um aluno. Se a média for maior ou igual a 7, o resultado será “aprovado”. E se a média for menor que 7 e maior ou igual a 5, o resultado será “recuperação”. Senão o resultado será “reprovado”.

Temos uma condição intermediária, essa condição **else if** permite a concatenação de diversos testes, podendo retornar “true” ou “false”. Conhecemos como condições aninhadas.

Exemplo:

```
if(fltMedia >= 7){
        System.out.println(“Aprovado”);
}else if(fltMedia < 7 && fltMedia >= 5){
        System.out.println(“Recuperação”);
}else{
        System.out.println(“Reprovado”);
}
```

Observe que o “else” não deverá possuir uma expressão lógica, pois o programa entenderá que se nenhuma expressão acima for “true”, deverá por padrão executar a instrução encontrada dentro de “else”.

Comando switch

Quem já conhece as linguagens C/C++ deve se lembrar que existe o comando condicional **switch**. Pois bem, em Java ele também existe, e o que esse comando faz é aceitar uma variável e compará-la a uma lista de constantes, executando uma linha de código (ou um bloco) quando encontrar uma correspondência.

Emprega-se extensivamente o comando **switch** na avaliação de uma sequência de testes. Esta, porém, não é tão legível e elegante, de forma que pode ser difícil seguir o fluxo do programa com ela.

Sintaxe básica:

```
switch(<variável>){
        case <constante1>:
                <instrução1>;
                break;
        case <constante2>:
                <instrução2>;
                break;
        default:
                <instrução3>;
}
```

Aqui temos que **<variável>** refere-se à variável cujo valor será testado contra as constantes **<constante1>**, **<constante2>**. Quando um dos testes for satisfeito, a linha de código será executada. O comando **break** é utilizado para informar que a sequência de comandos terminou. No caso de omissão, a execução dos comandos continuará no próximo **case**.

Exemplo:

```
switch(chrGenero){
        case ‘M’:
                System.out.println(“Masculino”);
                break;
        case ‘F’:
                System.out.println(“Feminino”);
                break;
        default:
                System.out.println(“Outro”);
}
```

Estrutura de repetição for

As estruturas de repetição, normalmente denominada *loops* ou laços, são também muito úteis na programação, pois permitem que um mesmo trecho de código seja executado várias vezes.

O comando **for** é composto por três seções: a **inicialização**, o **teste** e a **atualização**. Na inicialização, a variável (ou variáveis) que controla a quantidade de vezes que o *loop* será repetido recebe um valor.

Na seção de teste, essa mesma variável é comparada a uma constante ou a outra variável do programa. Enquanto esse teste resultar em “true”, o *loop* será repetido.

Na seção atualização, temos o comando que atualizará o valor existente na variável de controle. Todas essas seções são opcionais.

Sintaxe básica:

```
for(<variável>;<condição>;<atualização>){
        <instrução>;
}
```

Exemplo:

```
for(intContador = 1; intContador <= 10; intContador++){
        System.out.println(intContador+” x 7 ”+(intContador * 7));
}
```

TECH
soluções em ti

Outro exemplo:

```
inContador = 1;
for(;intContador <= 10;){
        System.out.println(intContador+” x 7 “+(intContador * 7));
        intContador++;
}
```

Note que apesar de omitirmos duas seções (**inicialização** e **atualização**), os sinais de ponto-e-vírgula permaneceram, para delimitar cada uma delas corretamente.

Estrutura de repetição while

A estrutura de repetição **while**, não possui uma seção para inicialização, outra para teste de expressão e uma terceira para a atualização da variável de controle. Ela apenas testa a expressão lógica e executa um comando ou bloco de instruções enquanto dela resultar um valor “true”.

A **inicialização** da variável que controla o bloco *loop* deve ser feita fora dele. Já a **atualização** é feita dentro do bloco de instruções.

Sintaxe básica:

```
while(<expressão>){
        <instrução>;
}
```

TECH
soluções em ti

Aqui, **<expressão>** é avaliada logo no início do *loop*, ou seja, antes da execução do comando ou bloco de instruções. Devido a esse fato, o comando pode não ser executado nem mesmo uma vez, já que a expressão pode retornar resultado “false” logo na primeira análise.

Exemplo:

```
int intContador;
intContador = 1;
while(intContador <= 10){
        System.out.println(intContador+” x 7 “+(intContador * 7));
        intContador++;
}
```

TECH
soluções em ti

Estrutura de repetição do/while

O último tipo de loop é o **do**/**while**, no qual um conjunto de instruções pode ser executado enquanto que uma expressão lógica que controla a repetição resulta em um valor “true”.

Como no *loop* **while**, a **inicialização** e a **atualização** da variável da expressão de controle devem ser separadas. O teste da condição é realizado no fim do bloco, após a execução dos comandos. Isso significa que eles serão executados ao menos uma vez.

Sintaxe básica:

```
do{
        <instruções>;
}(<expressão>);
```

TECH
soluções em ti

Exemplo:

```
int intContador;
intContador = 1;
do{
        System.out.println(intContador+” x 7 “+(intContador * 7));
        intContador++;
}while(intContador <= 10);
```

TECH
soluções em ti

Comandos break e continue

Já vimos que o comando **break** é utilizado para indicar o término de uma sequência de linhas de instruções em um **case** do comando **switch**.

Mas ele tem também outra função, que é interromper imediatamente a execução de um *loop*, fazendo com que o fluxo do programa prossiga na linha imediatamente seguinte ao fim deste *loop*, mesmo que após o comando **break** haja mais instruções.

Outro comando que afeta o comportamento de *loops* é **continue**. Embora seja parecido com o **break**, seu funcionamento é um pouco diferente. São parecidos no sentido em que pulam qualquer instrução que exista após eles.

No entanto, o **continue** é diferente porque não força a saída imediata da estrutura de repetição, mas faz com que seja executada novamente a avaliação – e, com isso, a próxima iteração será executada.

Exemplo break:

```
int intContador = 1;
while(intContador < = 6){
        if(intContador == 4){
                break;
        }
        System.out.print(intContador+” ”);
        intContador++;
}
```

Resultado: 1 2 3

Exemplo continue:

```
for(int intContador = 1; intContador <= 6; intContador++){
        if(intContador == 4){
                continue;
        }
        System.out.print(intContador+” “);
}
```

Resultado: 1 2 3 5 6

Vetores, Matrizes

Um vetor nada mais é do que um agrupamento de valores que possuem o mesmo tipo de dado. Essa coleção de valores é referenciada dentro do programa com um único nome, e cada elemento pode ser acessado por meio de um valor numérico que representa a sua posição dentro do conjunto todo.

Em Java, a declaração de um vetor é feita pela seguinte sintaxe:

tipo nome_vetor[ ] = new tipo[tamanho]

Em que **tipo** se refere ao tipo de dado que cada elemento do vetor deve armazenar. Outra forma de declarar um vetor é:

tipo[ ] nome_vetor = new tipo[tamanho];

Por exemplo, para declarar um vetor de vinte elementos do tipo *float* (ponto flutuante), devemos utilizar a sintaxe:

```
float fltVetor[ ] = new float[20];
```

Ou

```
float[ ] fltVetor = new float[20];
```

A indexação do vetor, ou seja, o acesso a cada elemento, é feita levando-se em consideração que o primeiro elemento do vetor possui índice 0.

No exemplo apresentado há somente uma dimensão, mas também podemos ter mais dimensões. Nesse caso, o vetor acaba sendo uma matriz. Para declarar uma matriz, a sintaxe é a mesma, com diferença que podemos especificar os diversos tamanhos mediante o uso de colchetes, da seguinte forma:

```
float[ ] fltMatriz = new float[5][20];
```

Para acessar um dos elementos da matriz, é necessário referenciar primeiro o índice da linha e depois o da coluna. Essa característica pressupõe que o índice à direita é alterado mais rapidamente que o índice da esquerda.

Exemplo vetor:

```
int intDia;
float fltValor, fltMedia;
float fltTemperatura[ ] = new float[5];
byte btValor[ ] = new byte[10];
```

Declaramos as variáveis acima, sendo que duas delas representam vetores.

```
System.out.println(“Media de Temperatura”);
System.out.println(“”);
for(intDia = 0; intDia < 5; intDia++){
        fltTemperatura[intDia] = 0;
```

```
try{
        for(intDia = 0; intDia < 5; intDia++){
                System.out.print("Digite a "+(intDia)+"a. temperatura:");
                System.in.read(btValor);
                String strBuffer = new String(btValor);
                String strValor = new String(strBuffer.trim());

                fltTemperatura[intDia] = Float.parseFloat(strValor);
        }
        fltMedia = (fltTemperatura[0] + fltTemperatura[1] +
fltTemperatura[2] + fltTemperatura[3] + fltTemperatura[4]) / 5;
        System.out.println("");
        System.out.print("A media das temperaturas e "+fltMedia);
```

```
        }catch(IOException ioException){
                ioException.printStackTrace();
        }
}
```

Esse código utiliza um vetor denominado **fltTemperatura** para armazenar as temperaturas digitadas pelo usuário. Os elementos desse vetor são inicializados com valor 0 e, então dentro de um laço **for** que varia de 0 a 4, ocorre a entrada das temperaturas. A média é obtida somando-se as cinco temperaturas e dividindo-se o total por cinco.

Note ainda que foi utilizada outra técnica para a entrada de dados pelo teclado: em vez de criarmos um objeto do tipo **Scanner**, utilizamos o método **read** da classe **System**. (pertence ao pacote java.io.*);

Procedimentos, Funções

Uma função representa um conjunto de ações que um programa deve executar. Criamos uma função quando desejamos que essas ações sejam repetidas em diversas partes do programa. Assim evitamos a repetição das mesmas linhas de código.

Sintaxe básica:

```
int nome_funcao(arg1, arg2, ...){
        escopo da função
}
```

A chave final representa o fim da função e, embora não esteja incluída no programa executável, faz com que haja um retorno à linha imediatamente seguinte à que efetuou a chamada da função.

Quando criamos uma função utilizando o operador **void**, significa que é sem retorno de valor (procedimento), mas, quando criamos uma função sem o operador **void**, significa que é com retorno de valor, e o operador **return** se torna obrigatório no corpo da função.

Exemplo procedimento:

```
void somaInteiros(int intX, int intY){
        System.out.println("Resultado da soma: "+(intX + intY));
}
```

Exemplo de função:

```
int somarInteiros(int intX, int intY){
        int intSoma = intX + intY;
        return intSoma;
}
```

Prosseguiremos no próximo slide...

Professor: Anderson Henrique

Programador nas Linguagens Java e PHP