Unidade 3: Estruturas de Controle

Parte 2 - Lógica, SWITCH, FOR, WHILE e DO WHILE

Prof. Daniel Caetano

Objetivo: Apresentar a composição lógica em estruturas de decisão e as estruturas FOR, WHILE e DO WHILE.

Bibliografia: DEITEL, 2005; CAELUM, 2010; HOFF, 1996.

INTRODUÇÃO

Nas aulas anteriores foram observadas as estruturas básicas de um programa, incluindo a estrutura de decisão com a instrução IF. Esta instrução foi apresentada com apenas uma comparação, sua forma mais comum. Nesta aula será apresentada a sintaxe que permite realizar diversas comparações com apenas uma estrutura IF. Para casos mais complexos, será apresentada a estrutura SWITCH.

Adicionalmente, serão apresentadas as estruturas de loop, isto é, de repetição, isto é, as estruturas FOR, WHILE e DO WHILE.

1. COMPARAÇÕES MÚLTIPLAS COM IF

Já vimos como fazer seleção de código com comparações múltiplas **aninhadas**, isto é, uma colocada dentro da outra. Para este efeito, foi usada a estrutura do tipo **if ~ else**. Resumidamente, a estrutura if ~ else serve para situações em que uma decisão simples é necessária, ou seja: existem duas possibilidades e nunca ambas ocorrem ao mesmo tempo: se um trecho de código executar, o outro não será executado.

Por exemplo: um aluno é aprovado se tiver média maior ou igual a 50. Então, a verificação é: "Se nota é maior ou igual a 50, aluno aprovado. Caso contrário, aluno reprovado". Em Java isso poderia ser expresso da seguinte forma:

```
if (notaDeProva >= 50) {   // se nota de prova maior ou igual a 50...
       aprovado = 1;       // aluno aprovado
       }
else   {                   // caso contrário...
       aprovado = 0;       // aluno não aprovado
       }
```

Como vimos, as estruturas **if~else** podem ser aninhadas, ou seja, podemos ter ifs dentro de ifs. Por exemplo: se o aluno passasse APENAS se tiver nota de prova maior que 50 e frequencia maior que 75, a estrutura ficaria assim:

```
if (notaDeProva < 50) {          // se nota de prova menor que 50...
      aprovado = 0;              // aluno reprovado
      }
else{                            // caso contrário... (nota de prova >= 50)
      if (frequencia >= 75) {    // se a frequencia maior ou igual a 75...
            aprovado = 1;        // aluno aprovado
            }
      else {                     // caso contrario (prova>=50 e freq<75)
            aprovado = 0;        // aluno não aprovado
            }
      }
```

Um outro jeito de conseguir o mesmo efeito é com associação lógica de comparações, usando as operações lógicas clássicas E (&&), OU (||) e NÃO (!) dentro do IF. Por exemplo, o código anterior pode ser simplificado assim:

```
// se nota de prova menor que 50 ou frequencia menor que 75...
if (notaDeProva < 50 || frequencia < 75 ) {
      aprovado = 0;              // aluno reprovado
      }
else{                            // caso contrário...
      aprovado = 1;       // aluno aprovado
      }
```

Embora um pouco mais complexa à primeira vista, esta notação facilita a leitura, reduz o código e pode tornar o programa mais eficiente. Lembrando a tabela verdade das operações lógicas:

A	Operação	B	Resultado
FALSO	*OU (||)*	FALSO	FALSO
FALSO	*OU (||)*	VERDADEIRO	VERDADEIRO
VERDADEIRO	*OU (||)*	FALSO	VERDADEIRO
VERDADEIRO	*OU (||)*	VERDADEIRO	VERDADEIRO
FALSO	*E (&&)*	FALSO	FALSO
FALSO	*E (&&)*	VERDADEIRO	FALSO
VERDADEIRO	*E (&&)*	FALSO	FALSO
VERDADEIRO	*E (&&)*	VERDADEIRO	VERDADEIRO
-	*NÃO (!)*	FALSO	VERDADEIRO
-	*NÃO (!)*	VERDADEIRO	FALSO

2. SELEÇÃO COM SWITCH~CASE

A estrutura do tipo switch~case existe para situações em que temos um número finito de possibilidades de tarefas a executar, dependendo de um único valor. Por exemplo, se um programa imprime o estado atual de um MP3 player e os estados possíveis são:

0- parado
1- tocando,
2- pausa
3- erro

Isso poderia ser feito com um switch na seguinte forma:

```
switch(estadoDoMP3) {
	case 0:
		System.out.println("MP3 parado");
		break;
	case 1:
		System.out.println("MP3 tocando");
		break;
	case 2:
		System.out.println("MP3 em pausa");
			break;
	case 3:
		System.out.println("Erro na leitura do MP3");
		break;
	default:
		System.out.println("Erro desconhecido");
		break;
	}
```

Note que para cada valor de estado, um determinado "case" será executado. A instrução break faz com que a execução pule para a primeira linha após os delimitadores { } do switch. Caso não se use a instrução break, a execução continua com o caso seguinte, até encontrar um break.

Note também o caso especial "default". Embora nem sempre estritamente necessário, é uma boa prática de programação usá-lo, pois ajuda a diagnosticar problemas de lógica no software. No exemplo acima, qualquer estadoDoMP3 diferente de 0 a 3 causará a execução do caso "default". Como não é possível fazer a menor idéia do que isso seja (já que o valor do estadoDoMP3 só deveria ser de 0 a 3) este caso especial imprime uma mensagem dizendo que um erro desconhecido ocorreu.

3. INSTRUÇÕES DE REPETIÇÃO

O Java fornece uma número mais que suficiente de instruções de repetição, que podem ser **aninhadas**, isto é, uma colocada dentro da outra. Estas instruções são as de construção do tipo **for**, **while** e **do ~ while**. Via de regra é possível substituir uma delas pela outra, mas há casos em que é mais cômodo usar uma ou outra, em favor da legibilidade.

3.1. Estrutura FOR

A instrução for é usada quando é necessário realizar uma tarefa por um número determinado de vezes. Ela compreende 4 partes: uma inicialização, um teste de finalização, uma descrição de incremento e, finalmente, o trecho que deve ser repetido.

Por exemplo: se for necessário imprimir um determinado texto 3 vezes, podemos usar uma estrutura for da seguinte forma:

```
for (int i = 0; i < 3; i = i + 1) {
       System.out.printf ("Texto número %d \n", i);
       }
```

O Java lê este trecho de código como:

Repita o trecho de código entre {} respeitando as seguintes regras:

1) Faça i (o contador) valer 0
2) Verifique se i é menor que 3. Se sim, execute passo 3. Se não, termine o for.
3) Execute o trecho entre { }
4) Faça i = i + 1 (ou seja, some 1 ao contador)
5) Volta ao passo 2.

O resultado desta estrutura (freqüentemente chamada de *loop* ou *laço*) é:

Texto número 0
Texto número 1
Texto número 2

3.2. Estrutura WHILE

A instrução while é usada quando queremos que um dado conjunto de instruções seja executado até que uma dada situação ocorra. A instrução while compreende 2 partes: um teste de finalização e o trecho que deve ser repetido.

Por exemplo: se for necessário imprimir um determinado texto até que o usuário digite 0 como entrada, podemos usar uma estrutura while da seguinte forma:

```
Scanner input = new Scanner(System.in);
int dado = -1;

while (dado != 0) {
       System.out.println ("Digite o número zero!");
       dado = input.nextInt();
       }
```

O Java lê este trecho de código como "Repita o trecho de código entre {} respeitando as seguintes regras:"

1) Verifique se dado é diferente de 0. Se sim, execute passo 2. Se não, fim do while.
2) Execute o trecho entre { }
3) Volta ao passo 1.

Note que agora a inicialização da variável e a atualização da mesma, ao contrário do que normalmente acontece com o for, **não é** responsabilidade da estrutura while, e sim do código que está antes do *loop* e do código do *loop* respectivamente.

Note também que, se a variável **dado** for inicializada com o valor zero (ao invés de -1, como foi no exemplo), o trecho de código entre { } no while não será executado nenhuma vez, pois o teste é feito antes de qualquer coisa ser executado.

3.3. Estrutura DO~WHILE

A instrução do~while é usada quando é desejado que um dado conjunto de instruções seja executado até que uma dada situação ocorra, mas é necessário garantir que o conjunto de instruções seja executado **pelo menos uma vez**. A instrução do~while compreende 2 partes: um trecho que deve ser repetido e um teste de finalização.

Usando o mesmo exemplo da instrução while, se for necessário imprimir um determinado texto até que o usuário digite 0 como entrada, podemos usar uma estrutura do~while da seguinte forma:

```
Scanner input = new Scanner(System.in);

do {
      System.out.println ("Digite o número zero!");
      dado = input.nextInt();
      } while (dado != 0);
```

O Java lê este trecho de código como: "Repita o trecho de código entre {} respeitando as seguintes regras:"

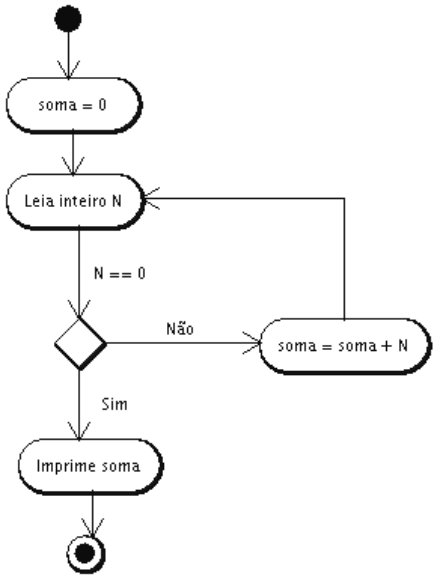
1) Execute o trecho entre { }
2) Verifique se dado é diferente de 0. Se sim, volte ao passo 1. Se não, termine o do-while.

Note que neste caso não foi necessário inicializar a variável, já que ela é lida dentro do *loop* antes de ser testada. De qualquer forma, assim como no caso do while, no do~while a inicialização da variável e a atualização da mesma, **não é** responsabilidade da estrutura do~while, e sim do código que está antes do *loop* e do código do *loop* respectivamente.

Note também que, independente do valor inicial da variável **dado**, o trecho de código entre { } no do~while será executado **pelo menos uma vez** já que o teste só é feito depois que o conteúdo do *loop* tiver sido executado uma vez.

4. IMPLEMENTANDO CÓDIGO

Vamos realizar um programa que leia números inteiros do teclado até que o número zero seja digitado. A cada número digitado, o resultado é armazenado na variável **soma**. O diagrama está a seguir:



While.java

```
import java.util.*;

class While {
   public static void main(String[] args) {
      int N;
      int soma;
      Scanner teclado = new Scanner(System.in);
      soma = 0;

      do {
         // Lê número N
         System.out.print("Digite um número inteiro ");
         System.out.print("ou 0 para finalizar: ");
         N = teclado.nextInt();
         if (N != 0) {
            soma = soma + N;
            }
         } while (N != 0);

      // Executa se o valor estiver na faixa esperada!
      System.out.println ("A soma final é: " + soma);
      }
   }
```

5. EXERCÍCIOS

A) Imprima todos os números de 100 a 220.
B) Imprima a soma dos números de 1 a 300.
C) Imprima todos os múltiplos de 7 entre 1 e 200.
D) Imprima os fatoriais dos números 1 a 10.

N! = N*(N-1)!
Ex: 1! = 1
2! = 2*1 = 2
3! = 3*2*1 = 6
4! = 4*3*2*1 = 24
...

E) Se quisermos fazer o fatorial de 1 a 40, o que acontece com a velocidade? E com os resultados? Altere o programa com variáveis do tipo **long** para corrigir os resultados.

6. BIBLIOGRAFIA

CAELUM, FJ-11: Java e Orientação a Objetos. Acessado em: 10/01/2010. Disponível em: < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos/ >

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

HOFF, A; SHAIO, S; STARBUCK, O. **Ligado em Java**. São Paulo: Makron Books, 1996.