VOLTAR

Estruturas de Controle de Fluxo e Laço em Java

Tolas as linguagens de programação modernas (como o Java) implementam algo que chamamos de controle de fluxo e laços de repetição. Aprenderemos, aqui, todos os diferentes tipos de controle de fluxo e laços que podem ser descritos em Java. Você sairá dessa aula sendo capaz de criar um programa um pouco mais refinado.

NESTE TÓPICO

Controle de fluxo

Podemos chamar o controle de fluxo, também, de estrutura de seleção. Em Java podemos (e devemos) usar estruturas de seleção para testes lógicos. Nas estruturas de seleção um determinado trecho de código é executado quando o teste lógico é verdadeiro e outro caso o resultado seja falso. Estamos falando do importante "if.. else.." (se, senão, em inglês), porém, há mais de uma forma de trabalharmos com estruturas de seleção em Java.

if..else

O if..else é a forma mais simples de fazermos o controle de fluxo de execução do programa, pois apenas desviamos a execução para determinada linha quando obtemos verdadeiro no teste ou para linha do else quando obtemos falso.

A estrutura do ***if..else,*** em Java, é:

```
//...
      if (<condição lógica>) {
            // trecho de código que será executado se o resultado for verdadeiro
      } else {
            //trecho de código que será executado se o resultado for falso
      }
//...
```

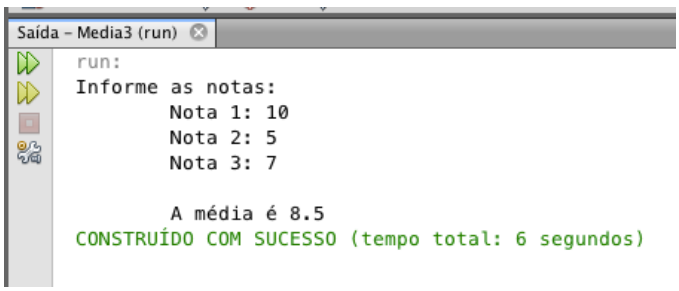
Podemos, também, ter mais de um else caso a segunda condição também não seja atendida, da seguinte forma (Teruel):

```
//...
      if (expressao 1) {
          //procedimento caso o resultado da expressao seja true
      } else if (expressao 2) {
           //procedimento caso o resultado da primeira expressa?o tenha sido false e da expressao atual tenha sido true.
      } else if (expressao 3){
           //procedimento caso o resultado da primeira e segunda expressoes tenham sido false e da expressao atual = true.
      } else {
           //procedimento caso o resultado de todas as expressoes anteriores tenha sido false
      }
//...
```

Para exemplificar, vamos programar um controle de fluxo que calcula a média de três notas, contudo pega **apenas as duas maiores** para o cálculo (Teruel):

```java
import java.util.Scanner;

public class Media3 {

    public static void main(String[] args) {
        //Declaracoes
        Scanner teclado = new Scanner(System.in);
        float nota1, nota2, nota3, media;

        //Leitura das notas
        System.out.println("Informe as notas: ");

        //Nota 1:
        System.out.print("\n\tNota 1: ");
        nota1 = teclado.nextFloat();

        //Nota 2:
        System.out.print("\tNota 2: ");
        nota2 = teclado.nextFloat();

        //Nota 3
        System.out.print("\tNota 3: ");
        nota3 = teclado.nextFloat();

        //Calcula usando um método com retorno:
        media = calculaMedia(nota1, nota2, nota3);

        System.out.println("\n\tA média é " + media);
    }

    //O método estático (static) permite ser invocado sem a instância da classe:
    public static float calculaMedia(float n1, float n2, float n3) {

        float media;

        //Verifica as duas maiores e tira a média:
        if (n1 < n2 && n1 < n3) {
            media = (n2 + n3) / 2;
        } else if (n2 < n3) {
            media = (n1 + n3) / 2;
        } else {
            media = (n1 + n2) / 2;
        }
        return media;
    }
}
```

Note que neste exemplo, foi criado um método estático para o cálculo da média. Métodos estáticos não precisam ter a instância da classe em memória para serem invocados (chamados), ou seja, podem ser chamados diretamente pelo seu nome. O resultado da execução deste código pode ser visto abaixo, como exemplo:



Resultado da execução - Média com dois maiores valores, de três

Comparação de String

Para comparar valores do tipo String, os operadores aritméticos convencionais não funcionam, pois trabalham apenas com valores numéricos e, como vimos anteriormente, Stings são cadeias de caracteres.

Para comparar valores String, então, utilizamos o método ***equals*** ou equalsIgnoreCase (quando não queremos que a caixa do texto seja levada em consideração). Veja um exemplo abaixo:

```
1.  public class ComparaString {
2.
3.      public static void main(String args[]) {
4.          //Declaracoes
5.          String nome1, nome2;
6.          Scanner tec = new Scanner(System.in);
7.
8.          //Leitura
9.          System.out.print("Informe o 1º nome: ");
10.         nome1 = tec.next();
11.
12.         System.out.print("Informe o 2º nome: ");
13.         nome2 = tec.next();
14.
15.         //compara deconsiderando a caixa:
16.         if (nome1.equalsIgnoreCase(nome2)) {
17.             System.out.println("\nOs nomes digitados são iguais!");
18.         } else {
19.             System.out.println("\nOs nomes digitados são diferentes!");
20.         }
21.     }
22. }
```

E os resultados podem ser:

```
run:
Informe o 1º nome: Josefino
Informe o 2º nome: JosefinA

Os nomes digitados são diferentes!
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Resultado 1 do código de comparação de Strings e Java

Ou, ainda:

```
run:
Informe o 1º nome: joSeFIno
Informe o 2º nome: JOSEFINO

Os nomes digitados são iguais!
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Resultado 2 do código de comparação de Strings e Java

Estrutura de seleção switch...case...default

Temos, em Java, uma outra forma de fazer controle de fluxo com encadeamento, que funciona exatamente igual a linguagem C: O Famoso *switch… case… default..*

Neste caso, usamos uma única variável para realizar comparações encadeadas. Sua estrutura é a seguinte:

```
//...
switch(veriavel) {
     case valor:
           //operacao a ser realizada se a veriavel contiver o valor especificado no case
          break;
      case valor2:
          // operacao a ser realizada se a veriavel contiver o valor especificado no case2
          break;
     case valor3:
           //operacao a ser realizada se a veriavel contiver o valor especificado no case3
          break;
     default:
           //operacaoa ser realizada se a veriavel não contiver o valor especificado em nenhum dos cases acima
}
//...
```

Vamos ver um exemplo de uso, bem simples, onde o usuário informa um valor e o programa verifica se o valor informado está sendo tratado (é previsto no código) ou não:

```
import java.util.Scanner;

public class SitchCase {

    public static void main(String[] args) {
        Scanner tec = new Scanner(System.in);

        System.out.print("Infomre um valor inteiro: ");
        int a = tec.nextInt();

        switch (a) {
            case 0:
                System.out.println("A variável recebeu o valor 0");
                break;
            case 1:
                System.out.println("A variável recebeu o valor 1");
                break;
            case 2:
                System.out.println("A variável recebeu o valor 2");
                break;
            default:
                 System.out.println("A variável recebeu qualquer outro valor não testado acima\n");
        }
    }
}
```

E os possíveis resultados dessa execução, são:

```
run:
Infomre um valor inteiro: 2
A variável recebeu o valor 2
CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
```

Resultado 1 do código de exemplo do switch case

Ou, ainda:

```
run:
Infomre um valor inteiro: 10
A variável recebeu qualquer outro valor não testado acima

CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Resultado 2 do código de exemplo do switch case

Para praticar, tente fazer um menu de seleção, onde é exibido ao usuário uma série de itens e, baseado na escolha do usuário, o programa realiza uma determinada tarefa, como uma calculadora, por exemplo.

Laços de repetição em Java

Antes de sairmos programando laços de repetição, é preciso ter certeza que você sabe o que é um laço de repetição e como ele funciona. Para isso vamos reforçar este assunto apresentando-lhe um importante vídeo que mostra em forma de pseudocódigo (português estruturado) o funcionamento do laço do tipo ?enquanto?:

Laços de repetição tipo *for*

Os laços do tipo *for* são equivalentes “para” em algoritmos, ou seja, “para i de um valor até outro, repita”. Os laços *for* permitem apenas comparações com valores numéricos para parada. Em Java, a sintaxe de laços *for* é:

```
//...
for(<condição de início> ; <condição de parada> ; <condição de incremento / decremento> {
     //… Trecho de código que será executado N vezes
}
//...
```

Para exemplificar, vamos fazer um programa que imprime todos os números pares entre 0 e 10. Neste caso, o laço ficará assim:

```
//...
for (int i = 0; i <= 10; i++){    //É o mesmo que: Para i de 0 a até 10 (inclusive), repita
     if (i % 2 == 0) // Verifica se a variável que é incrementada é divisível por 2, ou seja, se o resto de sua divisão por 2 é 0
          System.out.println(i);
}
//...
```

Note que no laço exemplificado acima, a variável de controle (i) foi declarada no próprio laço. Essa é uma prática comum em Java.

É importante ressaltar, também, que a variável de controle é incrementada ao final do laço.

O laço *for* funciona assim: A variável de controle a é inicializada com o valor inicial declarado na condição de início, o laço é executado. Depois disso a condição de parada é verificada. Se ela for falsa, então a variável de controle sofre o incremento ou decremento.

Para praticar, implemente um laço *for* que faça a impressão dos 50 primeiros números ímpares, contudo, de trás para frente, ou seja, 49, 47, 45 etc. Dica: Inicialize a variável de controle com 50 e use i-- para decrementar a variável de controle.

Laços de repetição tipo *while*

O laço do tipo *While* fornece ao programador uma liberdade maior de condições de incremento ou decremento pois ela pode ocorrer em qualquer momento dentro do laço. Outra diferença, segundo Teruel, é que laços do tipo *While* permite a comparação de valores não numéricos como condição de parada.

Diferentemente dos laços *for*, os laços *while* precisam que a variável de controle seja inicialiada antes. A sintaxe dos laços *while* e:

```
//...
while (<condição de parada>){
     //… Trecho de código que será executado N vezes
     <incremento ou decremento da variável de controle>
}
//...
```

Para exemplificar, vamos criar um pequeno programa que imprime os 10 primeiros valores ímpares. O código ficará assim:

```
1.  //...
2.  int i = 0
3.  while(i <= 10){
4.        if (i % 2 != 0) //Verifica se o resto da divisão por 2 é zero. Se não for, entã
    o o número é impar
5.            System.out.println(i)
6.       i++; //Incrementa a variável de controle
7.  }
8.  //..
```

Laços de repetição *do..while*

A grande diferença entre este tipo de laço está no momento em que ele verifica a condição de parada. Em laços do tipo ***do..while*** a condição de parada é verificada ao final de cada execução do laço, ou seja. o laço é executado e depois é verificada a condição. Segundo Teruel, isso quer dizer que o laço ***do..while*** sempre será executado ao menos uma vez.

A sintaxe deste tipo de laço é:

```
1.  do{
2.       //… Trecho de código que será executado N vezes
3.       <incremento ou decremento da variável de controle>
4.  } while(<condição de parada>)
```

Assim como o laço ***while***, este também precisa que a variável de controle seja inicializada antes. Vamos ver um exemplo de um pequeno trecho de código que é capaz de imprimir os 10 primeiros números, de trás para frente:

```
1.  //...
2.  int i = 10;
3.  do {
4.       System.out.println(i); //Imprime a própria variável de controle
5.       i--;     //Decremanta a variável de controle
6.  }while (i != 0);  //Condição de parada
7.  //...
```

Mas quando devo usar cada tipo de laço?

Não se preocupe, muitos programadores ficam confusos na hora de escolher qual laço usar, ou seja, essa é uma pergunta muito comum!

E, na verdade, não existe uma resposta exata para essa pergunta, pois cada situação é única. Há situações em que qualquer um dos três laços podem ser usados e dependerá, apenas, do “gosto” do programador. Contudo, haverá situações em que um determinado tipo de laço será necessário.

Isso quer dizer, então, que a resposta para essa pergunta é: Depende da situação.

Analise a situação cuidadosamente. Veja em que momento a variável de controle deve ser inicializada e a que momento ele deve ser incrementada.

Resumo dessa aula

Nesta aula você aprendeu coisas mais avançadas para aplicar em Java, como:

- Controle de fluxo usando o if..else
- Encadeamento de controle de fluxo else
- Laços de repetição em Java
  - For
  - While
  - Do..while

Você já está pronto para criar programas ainda mais avançados agora. Pratique bastante. Bons estudos e boa programação!

Quiz

Exercício Final

Estruturas de Controle de Fluxo e Laço em Java

INICIAR

Referências

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman

Avalie este tópico

ANTERIOR
Tipos de dados em Java

Índice

PRÓXIMO
Arrays e Coleções de Dados

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

® Todos os direitos reservados

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)