Módulo 1

Introdução à Programação I

Lição 6

Estruturas de controle

Versão 1.0 - Jan/2007

Autor
Florence Tiu Balagtas

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Reginald Hutcherson
Rebecca Ong
John Paul Petines
Sang Shin
Raghavan Srinivas
Matthew Thompson

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:
1.Microsoft Windows XP Profissional SP2 ou superior
2.Mac OS X 10.4.5 ou superior
3.Red Hat Fedora Core 3
4.Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)
NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:
1.Microsoft Windows 2000 Profissional SP4
2.Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
3.Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

1.Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
2.Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações:
http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Nas lições anteriores, foram mostrados programas seqüenciais, onde as instruções foram executadas uma após a outra de forma fixa. Nesta lição, discutiremos estruturas de controle que permitem mudar a ordem na qual as instruções são executadas.

Ao final desta lição, o estudante será capaz de:

- Usar estruturas de controle de decisão (**if** e **switch**) que permitem a seleção de partes específicas do código para execução
- Usar estruturas de controle de repetição (**while**, **do-while** e **for**) que permitem a repetição da execução de partes específicas do código
- Usar declarações de interrupção (**break**, **continue** e **return**) que permitem o redirecionamento do fluxo do programa

2. Estruturas de controle de decisão

Estruturas de controle de decisão são instruções em linguagem Java que permitem que blocos específicos de código sejam escolhidos para serem executados, redirecionando determinadas partes do fluxo do programa.

2.1. Declaração if

A declaração **if** especifica que uma instrução ou bloco de instruções seja executado se, e somente se, uma expressão lógica for verdadeira.

A declaração **if** possui a seguinte forma:

```
if (expressão_lógica)
    instrução;
```

ou:

```
if (expressão_lógica) {
    instrução1;
    instrução2
    ...
}
```

onde, **expressão_lógica** representa uma expressão ou variável lógica.

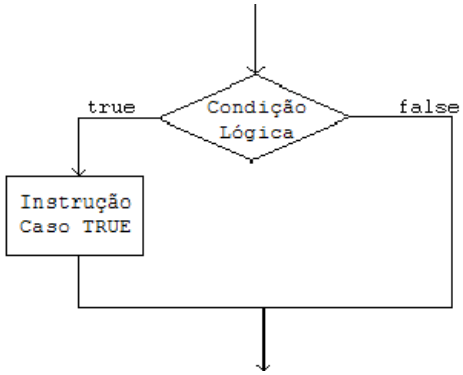

*Figura 1: Fluxograma da declaração **if***

Por exemplo, dado o trecho de código:

```
int grade = 68;
if (grade > 60) System.out.println("Congratulations!");
```

ou:

```
int grade = 68;
if (grade > 60) {
      System.out.println("Congratulations!");
      System.out.println("You passed!");
}
```

Dicas de programação:

1. Expressão lógica é uma declaração que possui um valor lógico. Isso significa que a execução desta expressão deve resultar em um valor true ou false.
2. Coloque as instruções de forma que elas façam parte do bloco **if**. Por exemplo:

```
if (expressão_lógica) {
    // instrução1;
    // instrução2;
}
```

2.2. Declaração if-else

A declaração **if-else** é usada quando queremos executar determinado conjunto de instruções se a condição for verdadeira e outro conjunto se a condição for falsa.

Possui a seguinte forma:

```
if (expressão_lógica)
    instrução_caso_verdadeiro;
else
    instrução_caso_falso;
```

Também podemos escrevê-la na forma abaixo:

```
if (expressão_lógica) {
    instrução_caso_verdadeiro1;
    instrução_caso_verdadeiro2;
    ...
} else {
    instrução_caso_falso1;
    instrução_caso_falso2;
    ...
}
```

Por exemplo, dado o trecho de código:

```
int grade = 68;
if (grade > 60)
    System.out.println("Congratulations! You passed!");
else
    System.out.println("Sorry you failed");
```

ou:

```
int grade = 68;
if (grade > 60) {
    System.out.print("Congratulations! ");
    System.out.println("You passed!");
} else {
    System.out.print("Sorry ");
    System.out.println("you failed");
}
```

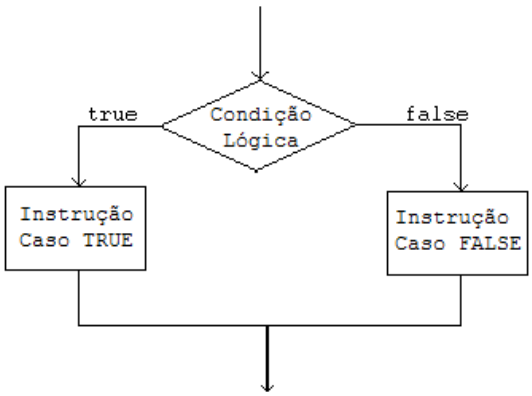

*Figura 2: Fluxograma da declaração **if-else***

Dicas de programação:

1. Para evitar confusão, sempre coloque a instrução ou instruções contidas no bloco **if** ou **if-else** entre chaves {}.
2. Pode-se ter declarações **if-else** dentro de declarações **if-else**, por exemplo:

```
if (expressão_lógica) {
    if (expressão_lógica) {
        ...
    } else {
        ...
    }
} else {
    ...
}
```

2.3. Declaração if-else-if

A declaração **else** pode conter outra estrutura **if-else**. Este cascateamento de estruturas permite ter decisões lógicas muito mais complexas.

A declaração **if-else-if** possui a seguinte forma:

```
if (expressão_lógica1)
    instrução1;
else if(expressão_lógica2)
    instrução2;
else
    instrução3;
```

Podemos ter várias estruturas **else-if** depois de uma declaração **if**. A estrutura **else** é opcional e pode ser omitida. No exemplo mostrado acima, se a **expressão_lógica1** é verdadeira, o programa executa a **instrução1** e salta as outras instruções. Caso contrário, se a **expressão_lógica1** é falsa, o fluxo de controle segue para a análise da **expressão_lógica2**. Se esta for verdadeira, o programa executa a **instrução2** e salta a **instrução3**. Caso contrário, se a **expressão_lógica2** é falsa, então a **instrução3** é executada.

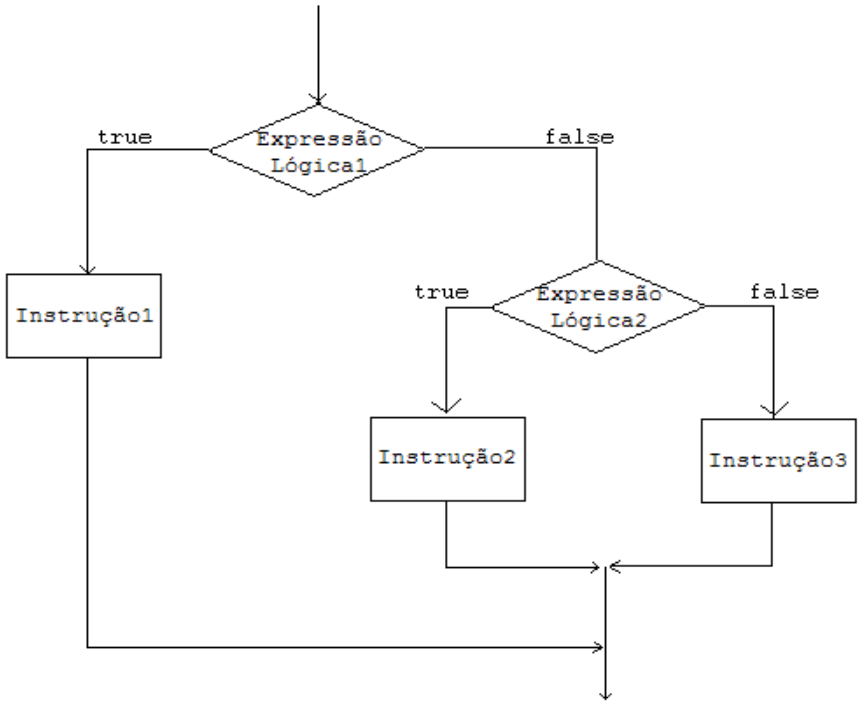

*Figura 3: Fluxograma da declaração **if-else-if***

Observe um exemplo da declaração **if-else-if** no seguinte trecho de código:

```
public class Grade {
    public static void main( String[] args ) {
        double grade = 92.0;
        if (grade >= 90) {
            System.out.println("Excellent!");
        } else if((grade < 90) && (grade >= 80)) {
            System.out.println("Good job!");
        } else if((grade < 80) && (grade >= 60)) {
            System.out.println("Study harder!");
        } else {
            System.out.println("Sorry, you failed.");
        }
    }
}
```

2.4. Erros comuns na utilização da declaração if

1. A condição na declaração **if** não avalia um valor lógico. Por exemplo:

```
// ERRADO
int number = 0;
if (number) {
    // algumas instruções aqui
}
```

a variável **number** não tem valor lógico.

2. Usar = (sinal de atribuição) em vez de == (sinal de igualdade) para comparação. Por exemplo:

```
// ERRADO
int number = 0;
if (number = 0) {
    // algumas instruções aqui
}
```

3. Escrever **elseif** em vez de **else if**.

```
// ERRADO
int number = 0;
if (number == 0) {
    // algumas instruções aqui
} elseif (number == 1) {
    // algumas instruções aqui
}
```

2.5. Declaração switch

Outra maneira de indicar uma condição é através de uma declaração **switch**. A construção **switch** permite que uma única variável inteira tenha múltiplas possibilidades de finalização.

A declaração **switch** possui a seguinte forma:

```
switch (variável_inteira) {
    case valor1:
        instrução1;     //
        instrução2;     // bloco 1
        ...       //
        break;
    case valor2:
        instrução1;     //
        instrução2;     // bloco 2
        ...       //
        break;
    default:
        instrução1;     //
        instrução2;     // bloco n
        ...       //
        break;
}
```

onde, **variável_inteira** é uma variável de tipo byte, short, char ou int. **valor1**, **valor2**, e assim por diante, são valores constantes que esta variável pode assumir.

Quando a declaração **switch** é encontrada, o fluxo de controle avalia inicialmente a **variável_inteira** e segue para o **case** que possui o valor igual ao da variável. O programa executa todas instruções a partir deste ponto, mesmo as do próximo **case**, até encontrar uma instrução **break**, que interromperá a execução do **switch**.

Se nenhum dos valores **case** for satisfeito, o bloco **default** será executado. Este é um bloco opcional. O bloco **default** não é obrigatório na declaração **switch**.

Notas:

1. Ao contrário da declaração **if**, múltiplas instruções são executadas sem a necessidade das chaves que determinam o início e término de bloco {}.
2. Quando um **case** for selecionado, todas as instruções vinculadas ao case serão executadas. Além disso, as instruções dos **case** seguintes também serão executadas.
3. Para prevenir que o programa execute instruções dos outros **case** subseqüentes, utilizamos a declaração **break** após a última instrução de cada **case**.

Dicas de Programação:

1. A decisão entre usar uma declaração **if** ou **switch** é subjetiva. O programador pode decidir com base na facilidade de entendimento do código, entre outros fatores.
2. Uma declaração **if** pode ser usada para decisões relacionadas a conjuntos, escalas de variáveis ou condições, enquanto que a declaração **switch** pode ser utilizada para situações que envolvam variável do tipo inteiro. Também é necessário que o valor de cada cláusula **case** seja único.

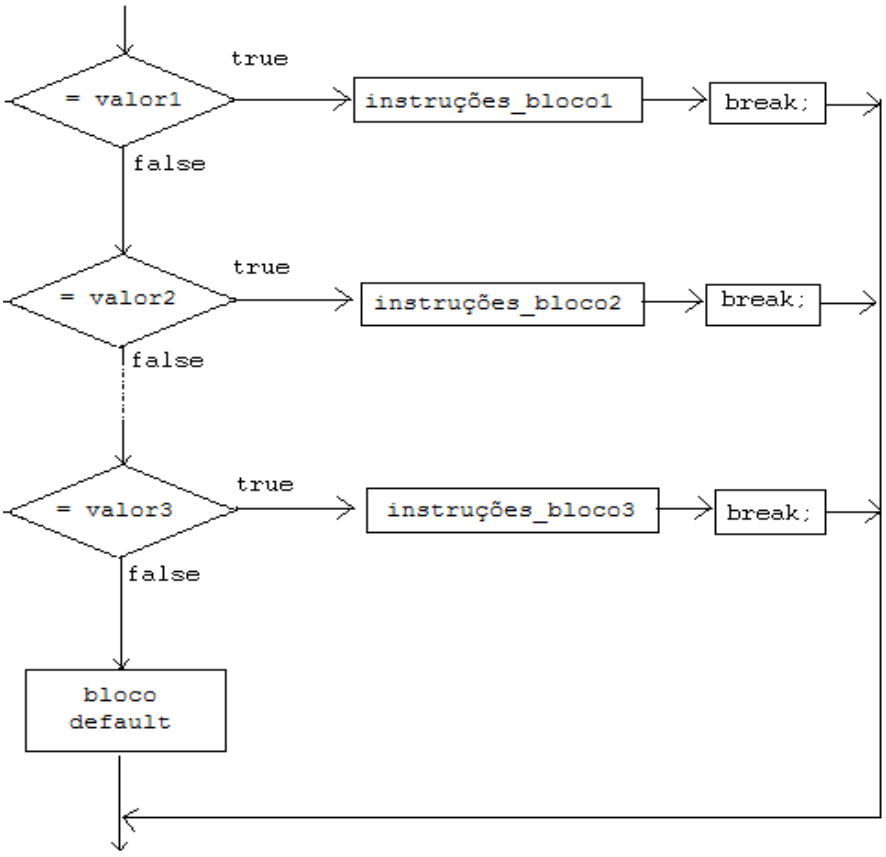

*Figura 4: Fluxograma da declaração **switch***

2.6. Exemplo para switch

```
public class Grade {
    public static void main(String[] args) {
```

```
                int grade = 92;
                switch(grade) {
                    case 100:
                        System.out.println("Excellent!");
                        break;
                 case 90:
                        System.out.println("Good job!");
                        break;
                    case 80:
                        System.out.println("Study harder!");
                        break;
                    default:
                        System.out.println("Sorry, you failed.");
                }
            }
        }
```

Compile e execute o programa acima e veremos que o resultado será:

```
        Sorry, you failed.
```

pois a variável **grade** possui o valor 92 e nenhuma das opções **case** atende a essa condição. Note que para o caso de intervalos a declaração **if-else-if** é mais indicada.

3. Estruturas de controle de repetição

Estruturas de controle de repetição são comandos em linguagem Java que permitem executar partes específicas do código determinada quantidade de vezes. Existem 3 tipos de estruturas de controle de repetição: **while**, **do-while** e **for**.

3.1. Declaração while

A declaração **while** executa repetidas vezes um bloco de instruções enquanto uma determinada condição lógica for verdadeira.

A declaração **while** possui a seguinte forma:

```
while (expressão_lógica) {
    instrução1;
    instrução2;
    ...
}
```

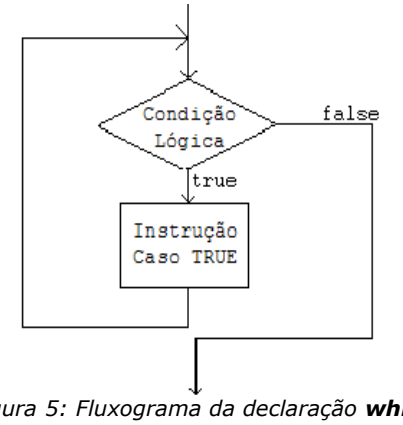

*Figura 5: Fluxograma da declaração **while***

As instruções contidas dentro do bloco **while** são executadas repetidas vezes enquanto o valor de **expressão_lógica** for verdadeira.

Por exemplo, dado o trecho de código:

```
int i = 4;
while (i > 0){
    System.out.print(i);
    i--;
}
```

O código acima irá imprimir 4321 na tela. Se a linha contendo a instrução **i--** for removida, teremos uma repetição infinita, ou seja, um código que não termina. Portanto, ao usar laços **while**, ou qualquer outra estrutura de controle de repetição, tenha a certeza de utilizar uma estrutura de repetição que encerre em algum momento.

Abaixo, temos outros exemplos de declarações **while**:

Exemplo 1:

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

```
        x++;
}
```

Exemplo 2:

```
// laço infinito
while (true)
    System.out.println("hello");
```

Exemplo 3:

```
// a instrução do laço não será executada
while (false)
    System.out.println("hello");
```

3.2. Declaração do-while

A declaração **do-while** é similar ao **while**. As instruções dentro do laço **do-while** serão executadas pelo menos uma vez.

A declaração **do-while** possui a seguinte forma:

```
do {
    instrução1;
    instrução2;
    ...
} while (expressão_lógica);
```

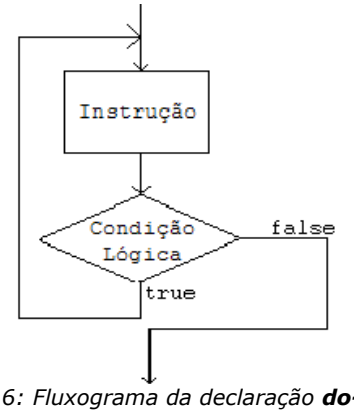

*Figura 6: Fluxograma da declaração **do-while***

Inicialmente, as instruções dentro do laço **do-while** são executadas. Então, a condição na **expressão_lógica** é avaliada. Se for verdadeira, as instruções dentro do laço **do-while** serão executadas novamente.

A diferença entre uma declaração **while** e **do-while** é que, no laço **while**, a avaliação da expressão lógica é feita antes de se executarem as instruções nele contidas enquanto que, no laço **do-while**, primeiro se executam as instruções e depois realiza-se a avaliação da expressão lógica, ou seja, as instruções dentro em um laço **do-while** são executadas pelo menos uma vez.

Abaixo, temos alguns exemplos que usam a declaração **do-while**:

Exemplo 1:

```
int x = 0;
```

```
do {
    System.out.println(x);
    x++;
} while (x<10);
```

Este exemplo terá 0123456789 escrito na tela.

Exemplo 2:

```
// laço infinito
do {
    System.out.println("hello");
} while(true);
```

Este exemplo mostrará a palavra **hello** escrita na tela infinitas vezes.

Exemplo 3:

```
// Um laço executado uma vez
do
    System.out.println(“hello”);
while (false);
```

Este exemplo mostrará a palavra **hello** escrita na tela uma única vez.

Dicas de programação:

1. Erro comum de programação ao utilizar o laço **do-while** é esquecer o ponto-e-vírgula (;) após a declaração **while**.

```
do {
    ...
} while (boolean_expression) // ERRADO -> faltou ;
```

2. Como visto para a declaração **while**, tenha certeza que a declaração **do-while** poderá terminar em algum momento.

3.3. Declaração for

A declaração **for**, como nas declarações anteriores, permite a execução do mesmo código uma quantidade determinada de vezes.

A declaração **for** possui a seguinte forma:

```
for (declaração_inicial; expressão_lógica; salto) {
    instrução1;
    instrução2;
    ...
}
```

onde:

declaração_inicial – inicializa uma variável para o laço
expressão_lógica – compara a variável do laço com um valor limite

salto – atualiza a variável do laço

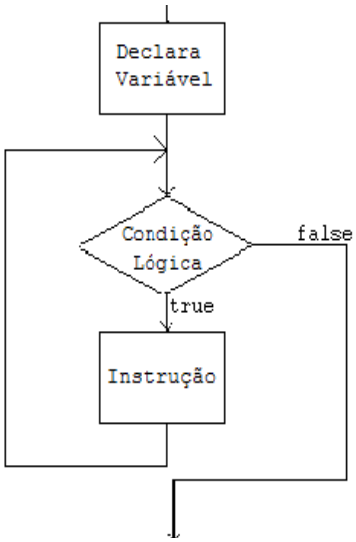

*Figura 7: Fluxograma da declaração **for***

Um exemplo para a declaração **for** é:

```
for (int i = 0; i < 10; i++) {
    System.out.print(i);
}
```

Neste exemplo, uma variável **i**, do tipo **int**, é inicializada com o valor zero. A expressão lógica "**i** é menor que 10" é avaliada. Se for verdadeira, então a instrução dentro do laço é executada. Após isso, a expressão **i** terá seu valor adicionado em 1 e, novamente, a condição lógica será avaliada. Este processo continuará até que a condição lógica tenha o valor falso.

Este mesmo exemplo, utilizando a declaração **while**, é mostrado abaixo:

```
int i = 0;
while (i < 10) {
    System.out.print(i);
    i++;
}
```

4. Declarações de Interrupção

Declarações de interrupção permitem que redirecionemos o fluxo de controle do programa. A linguagem Java possui três declarações de interrupção. São elas: **break**, **continue** e **return**.

4.1. Declaração break

A declaração **break** possui duas formas: **unlabeled** (não identificada - vimos esta forma com a declaração **switch**) e **labeled** (identificada).

4.1.1. Declaração unlabeled break

A forma **unlabeled** de uma declaração **break** encerra a execução de um **switch** e o fluxo de controle é transferido imediatamente para o final deste. Podemos também utilizar a forma para terminar declarações **for**, **while** ou **do-while**.

Por exemplo:

```
String names[] = {"Beah", "Bianca", "Lance", "Belle",
            "Nico", "Yza", "Gem", "Ethan"};
String searchName = "Yza";
boolean foundName = false;
for (int i=0; i < names.length; i++) {
    if (names[i].equals(searchName)) {
        foundName = true;
        break;
    }
}
if (foundName) {
    System.out.println(searchName + " found!");
} else {
    System.out.println(searchName + " not found.");
}
```

Neste exemplo, se a String "Yza" for encontrada, a declaração **for** será interrompida e o controle do programa será transferido para a próxima instrução abaixo da declaração **for**.

4.1.2. Declaração labeled break

A forma **labeled** de uma declaração **break** encerra o processamento de um laço que é identificado por um **label** especificado na declaração **break**.

Um **label**, em linguagem Java, é definido colocando-se um nome seguido de dois-pontos, como por exemplo:

```
teste:
```

esta linha indica que temos um **label** com o nome **teste**.

O programa a seguir realiza uma pesquisa de um determinado valor em um array bidimensional. Dois laços são criados para percorrer este array. Quando o valor é encontrado, um **labeled break** termina a execução do laço interno e retorna o controle para o laço mais externo.

```
int[][] numbers = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int searchNum = 5;
```

```
boolean foundNum = false;
searchLabel: for (int i=0; i<numbers.length; i++) {
    for (int j=0; j<numbers[i].length; j++) {
        if (searchNum == numbers[i][j]) {
            foundNum = true;
            break searchLabel;
        }
    } // final do laço j
} // final do laço i
if (foundNum) {
    System.out.println(searchNum + " found!");
} else {
    System.out.println(searchNum + " not found!");
}
```

A declaração **break**, ao terminar a declaração **for**, não transfere o controle do programa ao final de seu laço, controlado pela variável **j**. O controle do programa segue imediatamente para a declaração **for** marcada com o **label**, neste caso, interrompendo o laço controlado pela variável **i**.

4.2. Declaração continue

A declaração **continue** tem duas formas: **unlabeled** e **labeled**. Utilizamos uma declaração **continue** para saltar a repetição atual de declarações **for**, **while** ou **do-while**.

4.2.1. Declaração unlabeled continue

A forma **unlabeled** salta as instruções restantes de um laço e avalia novamente a expressão lógica que o controla.

O exemplo seguinte conta o a quantidade de vezes que a expressão "Beah" aparece no array.

```
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int count = 0;
for (int i=0; i < names.length; i++) {
    if (!names[i].equals("Beah")) {
        continue; // retorna para a próxima condição
    }
    count++;
}
System.out.println(count + " Beahs in the list");
```

4.2.2. Declaração labeled continue

A forma **labeled** da declaração **continue** interrompe a repetição atual de um laço e salta para a repetição exterior marcada com o **label** indicado.

```
outerLoop: for (int i=0; i<5; i++) {
    for (int j=0; j<5; j++) {
        System.out.println("Inside for(j) loop"); // mensagem1
        if (j == 2)
            continue outerLoop;
    }
    System.out.println("Inside for(i) loop"); // mensagem2
}
```

Neste exemplo, a mensagem 2 nunca será mostrada, pois a declaração **continue outerloop** interromperá este laço cada vez que **j** atingir o valor 2 do laço interno.

4.3. Declaração return

A declaração **return** é utilizada para sair de um método. O fluxo de controle retorna para a declaração que segue a chamada do método original. A declaração de retorno possui dois modos: o que retorna um valor e o que não retorna nada.

Para retornar um valor, escreva o valor (ou uma expressão que calcula este valor) depois da palavra chave **return**. Por exemplo:

```
return ++count;
```

ou

```
return "Hello";
```

Os dados são processados e o valor é devolvido de acordo com o tipo de dado do método. Quando um método não tem valor de retorno, deve ser declarado como **void**. Use a forma de **return** que não devolve um valor. Por exemplo:

```
return;
```

Abordaremos as declarações **return** nas próximas lições, quando falarmos sobre métodos.

5. Exercícios

5.1. Notas

Obtenha do usuário três notas de exame e calcule a média dessas notas. Reproduza a média dos três exames. Junto com a média, mostre também um :-) no resultado se a média for maior ou igual a 60; caso contrário mostre :-(

Faça duas versões deste programa:

1. Use a classe BufferedReader (ou a classe Scanner) para obter as notas do usuário, e System.out para mostrar o resultado.
2. Use JOptionPane para obter as notas do usuário e para mostrar o resultado.

5.2. Número por Extenso

Solicite ao usuário para digitar um número, e mostre-o por extenso. Este número deverá variar entre 1 e 10. Se o usuário introduzir um número que não está neste intervalo, mostre: "número inválido".

Faça duas versões deste programa**:**

1. Use uma declaração **if-else-if** para resolver este problema
2. Use uma declaração **switch** para resolver este problema

5.3. Cem vezes

Crie um programa que mostre seu nome cem vezes. Faça três versões deste programa:

1. Use uma declaração **while** para resolver este problema
2. Use uma declaração **do-while** para resolver este problema
3. Use uma declaração **for** para resolver este problema

5.4. Potências

Receba como entrada um número e um expoente. Calcule este número elevado ao expoente. Faça três versões deste programa:

1. Use uma declaração **while** para resolver este problema
2. Use uma declaração **do-while** para resolver este problema
3. Use uma declaração **for** para resolver este problema

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.