

Capítulo – 09

9.1 A estrutura switch-case

A estrutura **switch-case** refere-se a outra modalidade de desvio da execução do programa de acordo com certas condições, semelhante ao uso da instrução **if**. Ao trabalhar com uma grande quantidade de desvios condicionais contendo a estrutura **if-else**, pode-se comprometer a inteligibilidade do programa, dificultando sua interpretação. A estrutura **switch-case** possibilita uma forma **mais adequada e eficiente** de atender a esse tipo de situação, constituindo-se uma estrutura de controle com múltipla escolha.

A estrutura **switch-case** equivale a um conjunto de instruções **if** encadeadas, deixando mais claro o código. Veja abaixo a sua sintaxe:

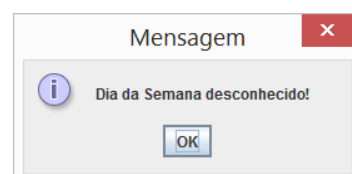
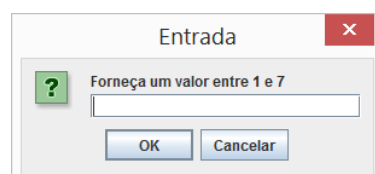
```
switch (<expressão>){  
    case 0: instruções; break;  
    case 1: instruções; break;  
    case 2: instruções; break;  
    case 3: instruções; break;  
    case 4: instruções; break;  
    default: instruções  
}
```

Opcional. Caso o case não encontre nenhum valor contido em <expressão> o default então é executado.

Na primeira linha deste **switch** é avaliado o resultado da expressão, que é comparado nas diretivas **case**, executando o bloco de instruções quando a expressão coincidir com o valor colocado ao lado direito do **case**. Em outras palavras, supondo que o valor da expressão seja igual a 2, serão executadas as **instruções localizadas entre case 2: e break**. A cada **case** o programa compara o valor da expressão com o valor colocado no **case**. Caso os valores sejam iguais, todas as instruções são executadas até que se encontre uma instrução **break**, que encerra o **switch** e faz a execução do programa desviar para outro ponto após a chave de encerramento do **switch**. O programa percorre todas as diretivas **case** até que uma delas seja igual à expressão inserida no **switch**. Caso nenhuma diretiva **case** possua o valor correspondente da expressão, serão executadas as instruções localizadas na diretiva **default** que é opcional.

Na classe abaixo veremos a forma clara da utilização da estrutura **switch-case**, simulando os dias da semana, em que o usuário entra com um número e o programa retorna o dia correspondente por extenso.

```
1 import javax.swing.JOptionPane;  
2 public class SwitchCase {  
3  
4     public static void main(String[] args) {  
5         String diaDaSemana = JOptionPane.showInputDialog("Forneça um valor entre 1 e 7");  
6  
7         if (diaDaSemana != null) {  
8             try {  
9                 int dia = Integer.parseInt(diaDaSemana);  
10                String extenso = "";  
11                switch (dia) {  
12                    case 1: extenso = "Domingo"; break;  
13                    case 2: extenso = "Segunda"; break;  
14                    case 3: extenso = "Terça"; break;  
15                    case 4: extenso = "Quarta"; break;  
16                    case 5: extenso = "Quinta"; break;  
17                    case 6: extenso = "Sexta"; break;  
18                    case 7: extenso = "Sábado"; break;  
19                    default : extenso = "Dia da Semana desconhecido!";  
20                }  
21                JOptionPane.showMessageDialog(null, extenso);  
22            }  
23            catch (NumberFormatException erro) {  
24                JOptionPane.showMessageDialog(null, "Entre apenas com valores numéricos!\n"  
25                    + erro.toString());  
26            }  
27        } else {  
28            JOptionPane.showMessageDialog(null, "Cancelando sistema...");  
29            System.exit(0);  
30        }  
31    }  
32 }
```



9.2 Novidades do Java 7 - Switch com String

Uma das novidades adicionadas ao Java desde sua versão 7 é a criação da estrutura switch com suporte para Strings. Até a versão 6, as estruturas switch-case suportavam apenas os tipos primitivos: byte, short, char, int, etc. Veja no trecho de código seguinte como podemos usar um valor String para testar uma condição na estrutura switch-case

```
2 import javax.swing.JOptionPane;
3
4 public class NovoSwitch {
5
6     public static void main(String[] args) {
7         String so = JOptionPane.showInputDialog(null, "Digite o nome do seu sistema operacional");
8
9         switch (so) {
10             case "Windows": {
11                 JOptionPane.showMessageDialog(null, "O SO Windows foi criado por Bill Gates.");
12                 break;
13             }
14             case "Linux": {
15                 JOptionPane.showMessageDialog(null, "O SO Linux foi criado por Linux Torvalds.");
16                 break;
17             }
18             case "Android": {
19                 JOptionPane.showMessageDialog(null, "O SO Android foi criado pela Google");
20                 break;
21             }
22             default: {
23                 System.out.println("Sistema Operacional desconhecido. ");
24             }
25         }
26     }
27 }
28 }
```

Uma observação interessante no switch é a opção de não colocação da palavra break ao final de cada condição case. Observe que o switch acima verifica se a String so contém “Windows” e, se por ventura este for o valor de so a instrução break orienta ao switch parar com as comparações e continuar o programa.

Mas se desejássemos que o switch comparasse **categorias** como no exemplo abaixo, onde na variável **tecnologia** que é recebida do usuário se refere a uma linguagem de programação ou um banco de dados. Neste o switch compara o valor recebido em tecnologia inicialmente com “java”, depois com c++, e por fim “cobol”. Se em tecnologia tivermos um destes três, a mensagem “Linguagem de Programação” será exibida e o switch será finalizado. Porém se a comparação não encontrar “java” ou c++ ou “cobol” imediatamente o switch analisará se o valor comparado é igual a “oracle” ou “sqlserver” ou ainda “postgresql”.

Caso o valor da variável não coincida com uma das opções então a mensagem **Tecnologia desconhecida** contida em default é exibida.

```
public class NovoSwitchSemBreak {

    public static void main(String[] args) {
        String tecnologia = "postgresql";

        switch (tecnologia){
            case "java":
            case "c++":
            case "cobol":
                System.out.println("Linguagem de Programação");
                break;
            case "oracle":
            case "sqlserver":
            case "postgresql":
                System.out.println("Banco de dados");
                break;
            default:
                System.out.println("Tecnologia desconhecida");
        }
    }
}
```

Execute esta classe e depois altere o valor da variável tecnologia para c++ e execute novamente.

Este tipo de switch é equivalente a:

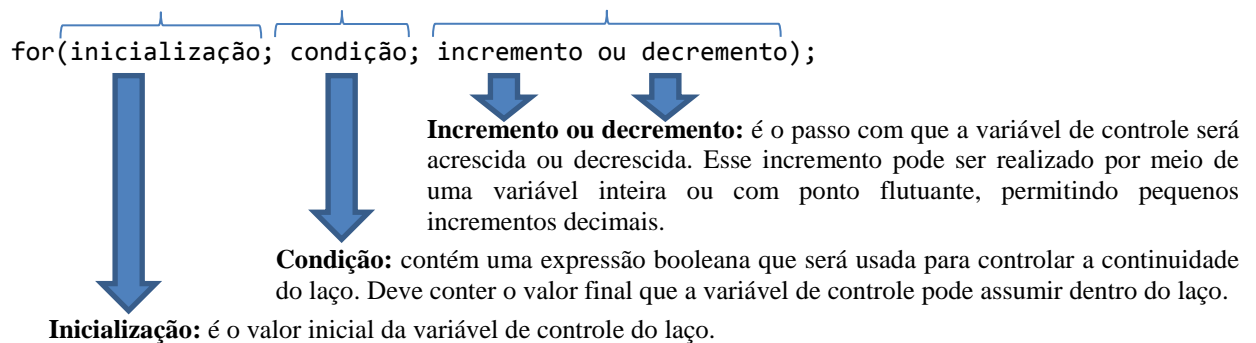
```
if (tecnologia.equals("java") || tecnologia.equals("c++") || tecnologia.equals("cobol"))
```

9.3 laços de repetição

Laços de repetição ou (looping) formam uma importante estrutura nas linguagens de programação por possibilitarem a repetição da execução de um bloco de instruções em um programa. Eles determinam que um certo bloco seja executado repetidamente até que a condição específica ocorra. A repetição é uma das estruturas mais usadas em programação, possibilitando a criação de contadores, temporizadores, rotinas para classificação, obtenção e recuperação de dados. A criação de laços de repetição em Java é feita a partir das estruturas `for`, `while` e `do-while`.

9.4 Uso do laço `for`

A instrução `for` é do tipo contador finito, isto é, ele realiza a contagem de um valor inicial determinando até um valor final determinando. Contém uma variável de controle do tipo contador, que pode ser crescente ou decrescente e possui a seguinte sintaxe:



Exemplo de um `for` que “roda” onze vezes.

```
for (int contador=0; contador <= 10; contador++){  
    <conjunto de instruções>;  
}
```

Dica! Perceba que a variável `contador` que inicia se em **0 zero** que iniciará todo o processo de voltas que o `for` vai realizar. Portanto o 0 zero já **será uma volta**, repetido todo o processo 11 vezes!

Isto pode ser interpretado como: inicialize a variável `contador` com zero e repita o conjunto de instruções enquanto o valor de `contador` for menor ou igual a 10. Cada vez que o conjunto de instruções é executado, o valor do contador é acrescido/incrementado (`contador++`). Observe que a variável `contador` pode ser declarada e inicializada na própria estrutura do `for`. Quando isso é feito a variável declarada tem um escopo(alcance) local, ou seja, ela “só vale” dentro do `for`. Após o encerramento do `for` a variável perde sua referência e não pode ser mais usada, salvo se declarada novamente. Mas se for declarada antes do `for` você poderá utilizar o valor contido nela.

Outros exemplos de laços de repetição usando `for`.

```
for(double x=5; x<=10; x=x+0.5)
```

Faz o valor de `x` variar de 5 até 10 com passo de 0.5, ou seja, 5.0, 5.5, 6.0, 6.5, ... 9.5, 10.0.

```
for(int x=10; x>=0; x=x-2)
```

Faz o valor de `x` variar de 10 até 0 com passo de -2, ou seja, 10, 8, 6, 4, 2, 0.

```
for(int x=a; x<=b; x++)
```

Faz o valor de `x` variar de `a` até `b` com passo de 1.

O exemplo abaixo mostra de forma bem simples o uso do for para fazer uma contagem progressiva de 0 até 9.

```
ContadorProgrsivo.java
1 package br.com.Capitulo9.LacoFor;
2
3 public class ContadorProgrsivo {
4
5     public static void main(String[] args) {
6         for (int i = 0; i < 10; i++) {
7             System.out.print(i + " ");
8         }
9         System.out.println("\nAcabou");
10    }
11 }
12
```

```
Console
<terminated> ContadorProgrsivo [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\javaw.exe (16/03/2015 16:22:52)
0 1 2 3 4 5 6 7 8 9
Acabou
```

O exemplo abaixo demonstra o uso do for na simulação de um relógio contador.

```
3 public class Relogio {
4
5     public static void main(String[] args) {
6         int horas, minutos, segundos;
7
8         for (horas=0;horas<24;horas++){
9             for (minutos=0;minutos<60;minutos++){
10                for (segundos=0;segundos<60;segundos++){
11
12                    System.out.println(horas + "h: " + minutos + "min: " + segundos + "s" );
13                    try
14                    {
15                        Thread.sleep(1000);
16                    }
17                    catch (InterruptedException erro)
18                    {
19                        //tratamento do erro
20                    }
21                }
22            }
23        }
24    }
25 }
26
27
```

Clique aqui para parar o relógio.

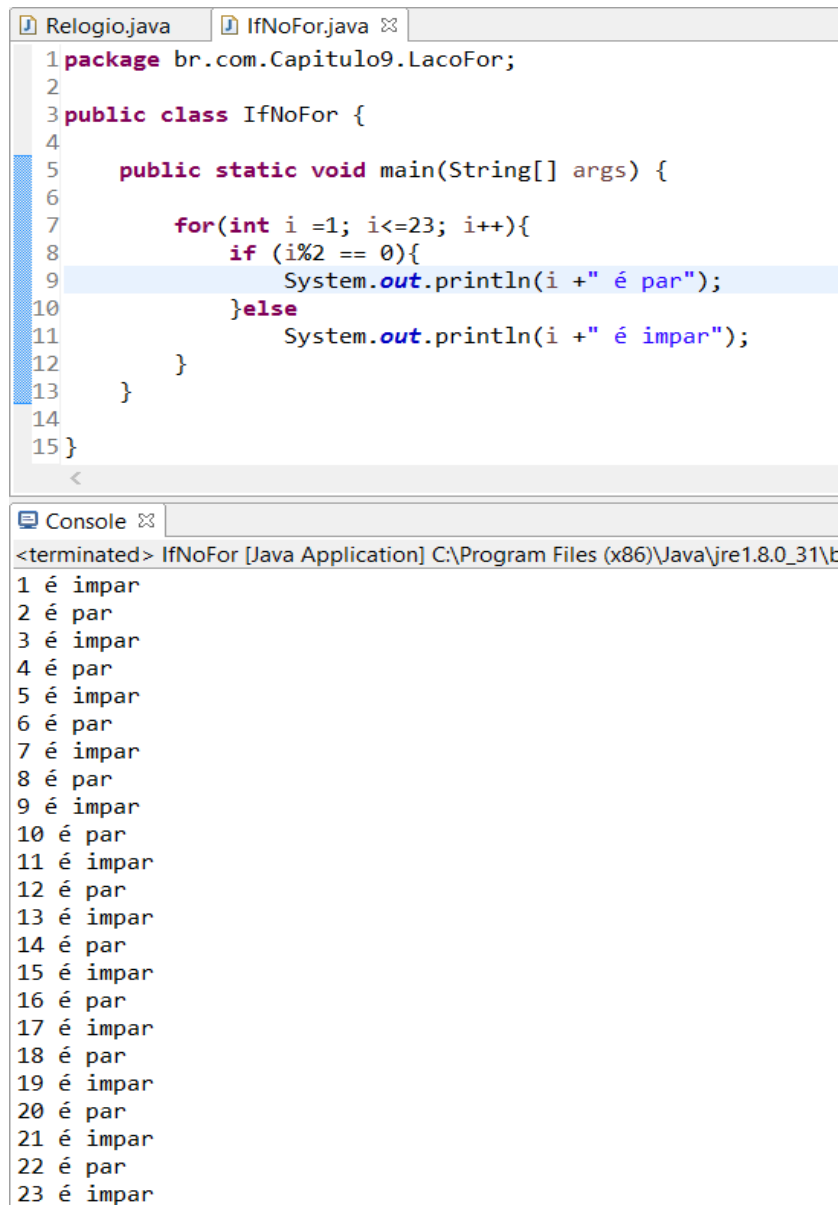
```
Console
Relogio [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\javaw.exe (16/03/2015 16:43:39)
0h: 0min: 0s
0h: 0min: 1s
0h: 0min: 2s
0h: 0min: 3s
0h: 0min: 4s
0h: 0min: 5s
0h: 0min: 6s
0h: 0min: 7s
```

Nas linhas 8 a 10 temos os laços de repetição para o controle das horas (0 a 23), minutos (0 a 59) e segundos (0 a 59). Um ponto a ser observado é a possibilidade de criação de um laço de repetição dentro do outro. Quando isso ocorre, o laço interno é executado n vezes, de acordo com o número de vezes definido pelo laço anterior. Como exemplo observe as linhas de 10 a 21. O laço mais interno (segundos) é executado 60 vezes a cada minuto (o laço superior). Da mesma forma o laço dos minutos será executado 60 vezes a cada hora (o laço superior)

Linha 15 contém um `Thread.sleep(1000);` que invoca um temporizador para esperar mil milissegundos (um segundo) até a repetição do for que controla os segundos. Esse procedimento pode gerar uma exceção, e por isso

deve ser usado no bloco `try-catch`. O uso de um `thread` talvez não seja a melhor forma de se criar uma temporização. Para isso existem outras classes como `Timer` e `TimerTask` cujo o estudo foge aos abordados neste curso.

9.5 – If dentro de um for



The screenshot shows an IDE with two tabs: `Relogio.java` and `IfNoFor.java`. The `IfNoFor.java` tab is active, displaying the following code:

```
1 package br.com.Capitulo9.LacoFor;
2
3 public class IfNoFor {
4
5     public static void main(String[] args) {
6
7         for(int i =1; i<=23; i++){
8             if (i%2 == 0){
9                 System.out.println(i + " é par");
10            }else
11                System.out.println(i + " é impar");
12        }
13    }
14
15 }
```

Below the code editor is a console window titled `Console`. It shows the output of the program:

```
<terminated> IfNoFor [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\t
1 é impar
2 é par
3 é impar
4 é par
5 é impar
6 é par
7 é impar
8 é par
9 é impar
10 é par
11 é impar
12 é par
13 é impar
14 é par
15 é impar
16 é par
17 é impar
18 é par
19 é impar
20 é par
21 é impar
22 é par
23 é impar
```

No exemplo acima utilizamos dentro do laço `for` um `if` que será repetido 22 vezes para avaliar o resto da divisão de cada um dos números que foram incrementados. Se o resto da divisão representado por “%” for zero isto significa que o número é par e a mensagem “é par” será exibida, caso contrário ou `else` a mensagem “é impar” aparecerá.

9.4 For com sentinela (controlando os 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.

```

3 class ForComSentinela {
4     public static void main(String[] args) {
5         int x = 1;
6         int y = 25;
7
8         for (int i = x; i < y; i++) {
9             if (i % 19 == 0) {
10                System.out.println("Achei um número divisível por 19 entre x e y " + i) ;
11                break;
12            }
13        }
14    }
15}

```

O código acima vai percorrer os números de x a y e 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 sem o atual ter efeito algum. Para isso usamos a palavra chave **continue**.

```

3 public class ForComContinue {
4
5     public static void main(String[] args) {
6         for (int i = 0; i < 100; i++) {
7
8             if (i > 50 && i < 60) {
9                 continue;
10            }
11            System.out.println(i) ;
12        }
13    }
14}

```

Quando i for maior que 50 e enquanto for < que 60 o loop pulará.

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

9.5 Laço de repetição while e do-While

```

while (condição) {
    <comandos>
}

```

Estas estruturas são muito semelhantes à estrutura for que vimos. A grande diferença é que elas apenas têm o critério de avaliação (**condição**). Caso esta expressão seja avaliada de forma positiva o bloco é executado até que esta condição se torne falsa. Por isso devemos ter cuidado com o temido While infinito. Sim pois o while não possui uma variável de controle como o for, sendo necessário o programador coloca-lo.

No caso acima nos <comandos> deveríamos ter algum trecho que em algum momento tornasse a condição do while falsa para que o laço seja quebrado.

Agora veremos o do-while:

```

do{
    <comandos>
}while (condição)

```

E na expressão do-while o bloco será executado pelo menos uma vez pois a (condição) é sempre avaliada depois da execução do bloco.

Vamos a um exemplo mais prático. Digite o seguinte código.

```
1
2
3 public class While {
4
5     public static void main(String[] args) {
6         int valor = 0;
7         while (valor < 10){
8             System.out.print(valor + " ");
9             valor = valor + 1;
10        }
11        System.out.println("\nAcabou");
12    }
13 }
14 }
```

Variável de controle, ela evita o while infinito.

Esta linha poderia ser substituída pelo operador de pós-incremento tente alterar para: `valor ++`

Console

<terminated> While [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\javaw.exe (16/03/2015 21:36:46)

0 1 2 3 4 5 6 7 8 9

Acabou

O exemplo acima produziu o mesmo efeito do primeiro exemplo no item 9.4 do `for`.

Veja agora um exemplo do mesmo programa, mas com o uso do `do-while`:

```
1 public class DoWhile {
2
3     public static void main(String[] args) {
4         int valor = 0;
5         do{
6             System.out.print(valor + " ");
7             valor ++;
8         }while (valor < 10);
9
10        System.out.println("\nAcabou");
11    }
12 }
13 }
14 }
```

Console

<terminated> DoWhile [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\

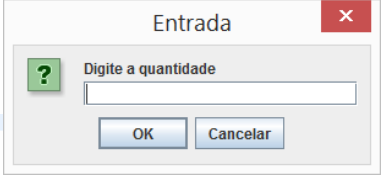
0 1 2 3 4 5 6 7 8 9

Acabou

Veja que o resultado foi o mesmo! A utilização do `while` ou `do-while` fica a critério do desenvolvedor, pois com as duas formas é possível chegar a resultados semelhantes. O uso de `while` será útil quando você não souber quantas vezes o loop será executado e quiser continuar a execução apenas enquanto alguma condição for verdadeira. Mas se você souber quantas vezes o loop será executado, um loop será mais simples.

Vamos aprender mais uma funcionalidade da estrutura `while` e `do-while`, mas agora utilizando as duas em uma só classe.

```
1 import javax.swing.JOptionPane;
2 public class ContadorWhile {
3
4     public static void main(String[] args) {
5         try{
6             int limite = Integer.parseInt(JOptionPane.showInputDialog("Digite a quantidade"));
7
8             int contador = limite;
9             while (contador >= 0) {
10                 System.out.println(contador);
11                 contador --;
12             }
13
14             System.out.println("Fim do contador regressivo\n");
15             contador = 0;
16
17             do{
18                 System.out.println(contador);
19                 contador++;
20             }while (contador <= limite);
21             System.out.println("Fim do contador progressivo");
22
23         }catch(NumberFormatException erro){
24             System.out.println("Não foi fornecido um número inteiro válido!\n" + erro.toString()); // se o argumento for inválido
25         }
26     }
27 }
28 }
```



```
<terminated> ContadorWhile [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\javaw.exe (16/03/2015 22:07:54)
3
2
1
0
Fim do contador regressivo
0
1
2
3
Fim do contador progressivo
```

Na linha 6 o usuário digita um número que será armazenado na variável `limite`. Caso ocorra algum erro de conversão, o direcionamento do programa é alterado para `NumberFormatException` (linha 23).

Na linha 8 o valor da variável `limite` é armazenado na variável `contador`. Isso foi realizado para manter o valor fornecido pelo usuário que será utilizado mais a frente.

Nas linhas de 9 a 12 uma vez que a variável `contador` receba um número inteiro positivo, é realizada sua contagem decrescente até o valor zero por meio da estrutura `while`. Caso seja fornecido um valor negativo, a contagem não é realizada pois a expressão `while` será falsa. A cada ciclo (chamado também de iteração) o valor da variável `contador` é impresso na tela (linha 10) e decrementado na (linha 11) até -1, quando o laço termina, pois -1 não é maior que zero.

Na linha 15 o valor da variável `contador` é zerado para que a contagem progressiva do próximo laço inicie do zero, já que até a linha anterior o valor da variável é -1.

Nas linha 17 a 20 temos um laço `do-while`. Como já dito, o `do-while` executa o bloco de instruções pelo menos uma vez e depois verifica a condição de sua expressão. No caso, o valor da variável `contador` vai de zero até o valor que o usuário forneceu inicialmente, armazenado na variável `limite`, portanto a contagem será progressiva.

9.6 Mantendo consistência de interface com `while` ou `for`.

Até este ponto os sistemas elaborados em Java não tinham a capacidade de retornar para a “tela” inicial, ou seja, o programa não era capaz de perguntar novamente as entradas que deveriam ser feitas pelo o usuário para processamento. Obrigando ao usuário ter que carregar todo o programa novamente caso este desejasse inserir uma nova sequência. Pense por exemplo, em um programa de notas escolares que recebe um nome de aluno e quatro notas, para então apresentar a média e um status como “aprovado” ou “reprovado”. Imagine ainda que você tenha 40 alunos de uma mesma sala de aula. Seria cansativo entrar com todas as informações para um mesmo aluno e

pós o processamento, o programa simplesmente fechasse obrigando ao usuário ter que carregar o programa e repetir todo o processo.

Para um melhor desempenho. Podemos nos utilizar dos laços de repetição com um **sentinela**, para que as perguntas que são feitas para o usuário através do JOptionPane possam ser apresentadas novamente para um novo processamento.

Ao invés de ter que programar repetidamente como no exemplo abaixo:

```
String aux1 = JOptionPane.showInputDialog(null, "Informe o 1ª nota: ");
String aux2 = JOptionPane.showInputDialog(null, "Informe o 2ª nota: ");
String aux3 = JOptionPane.showInputDialog(null, "Informe o 3ª nota: ");
String aux4 = JOptionPane.showInputDialog(null, "Informe o 4ª nota: ");
```

Poderíamos fazer:

```
do{
String nota=JOptionPane.showInputDialog(null, "Informe a nota " + (contador+1) + ": ou -1 para sair");
    contador = contador + 1;
}while (nota != -1);
```

Bem mais simples não? Claro que estas classes não estão completas, o objetivo foi mostrar o uso do while para que a mensagem apareça várias vezes na tela até ser digitado pelo usuário o valor (-1) onde o programa continuará o processamento.

Veja este exemplo:

```
1 import javax.swing.JOptionPane;
2
3 public class NotasWhileCmSentinela {
4     public static void main(String[] args) {
5         {
6             int total; // soma das notas
7             int contaNotas; // numero de notas inseridas
8             int nota; // valor da nota
9             double media; // número com ponto de fração decimal para a média
10
11             // fase de inicialização
12             total = 0;
13             contaNotas = 0;
14
15             // fase de processamento
16             nota = Integer.parseInt(JOptionPane.showInputDialog("Entre com a nota ou -1 para sair"));
17
18             while ( nota != -1) //faz um loop até ler o valor da sentinela -1 inserido pelo usuário
19             {
20                 total = total + nota; // adiciona nota ao total
21                 contaNotas = contaNotas + 1;
22
23                 nota = Integer.parseInt(JOptionPane.showInputDialog("Entre com a nota ou -1 para sair"));
24             }
25
26             if (contaNotas != 0 )
27             {
28                 // calcula a média de todas as notas inseridas
29                 media = (double) total / contaNotas;
30
31                 JOptionPane.showMessageDialog(null, "Numero total de notas é: "+ contaNotas + " "+ total +
32                 "\nTotal de "+contaNotas+ " notas inserida é "+ total +
33                 " A média da sala é " + media);
34             }
35
36             else
37                 JOptionPane.showMessageDialog(null, "Nenhuma nota foi inserida");
38         }
39     }
40 }
```

Agora construa esta classe e avalie o resultado

```
1 import javax.swing.JOptionPane;
2
3 public class Notas {
4
5     public static void main(String[] args) {
6         int contador = 0;
7         float nota, notaf;
8         boolean vezes = true;
9
10        float total = 0;
11        do{
12
13            notaf = Float.parseFloat(JOptionPane.showInputDialog(null, "Informe a nota " + (contador+1) + ": ou -1 para sair"));
14            if (notaf == -1){
15                break;
16            }
17            nota = notaf;
18            contador = contador + 1;
19            total = total + nota;
20
21        }while (vezes);
22
23        System.out.println("Total de notas lançadas " + contador + " A média é: " + total / contador);
24    }
25 }
```

10-Atividades

1) Desenvolva um programa que 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. Se não sabe o que é fatorial leia abaixo a explicação:

Faça um `for` que inicie uma variável `n` (número) como 1 e fatorial (resultado) como 1 e varia `n` de 1 até `int` `fatorial = 1`;

```
for (int n = 1; n <= 10; n++) {  
}
```

O fatorial de um número `n` é $n * n-1 * n-2 \dots$ até $n = 1$. Lembre-se de utilizar os parênteses.

Por exemplo, o fatorial de 4 é $1 \times 2 \times 3 \times 4 = 24$.

O fatorial de 0 é 1

O fatorial de 1 é $(0!) * 1 = 1$

O fatorial de 2 é $(1!) * 2 = 2$

O fatorial de 3 é $(2!) * 3 = 6$

O fatorial de 4 é $(3!) * 4 = 24$

5) No código do exercício anterior, aumente a quantidade de números que terão os fatoriais impressos, até 20, 30, 40. Em um determinado momento, além de o cálculo demorar, vai começar não a mostrar respostas, descubra o por que.

6) (opcional) Imprima a seguinte tabela, usando `for`s encadeados:

```
1  
2 4  
3 6 9  
4 8 12 16  
n n*2 n*3 .... n*n
```

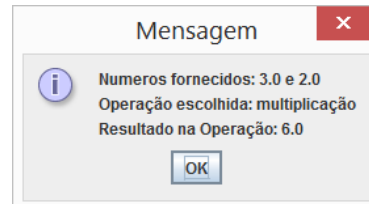
7) Usando JOptionPane, elabore uma classe que receba o valor de um produto e o código de desconto. O desconto deve ser calculado de acordo com o código fornecido na tabela abaixo;

Código	% Desconto
1	5
2	10
3	20
4	50

Então faça este exercício utilizando a estrutura switch-case, apresente em tela o novo valor do produto, depois de ser realizado o desconto, o programa deverá retornar a tela inicial após calcular o desconto informado para o produto. Caso o código não exista, deve ser emitida uma mensagem de aviso e novamente retornar a tela inicial.

8) Utilizando o switch-case elabore uma classe em que o usuário fornece dois números e uma letra correspondente à operação de acordo com a tabela abaixo.

Letra	Operação
A	Soma
B	Subtração
C	Multiplicação
D	Divisão



9) Faça uma classe que solicite uma senha, simulando um caixa eletrônico. Considere que a senha é Java. Caso o usuário forneça a senha correta, apresentar a mensagem “Senha Válida”; caso contrário, “Senha Inválida”. Se o usuário fornecer a senha incorreta três vezes seguidas, o programa deve apresentar a mensagem “Cartão Cancelado!”.

10) Faça uma classe que apresente em tela a soma de todos os números ímpares compreendidos entre 1 e 1000(1+3+5... +999);