

Tópico 3 – Programação Orientada a Objetos (POO)

Laços:

- while,
 - do..while,
 - for;
-
- Estruturas de erros;



Laços – *While* e *do while*

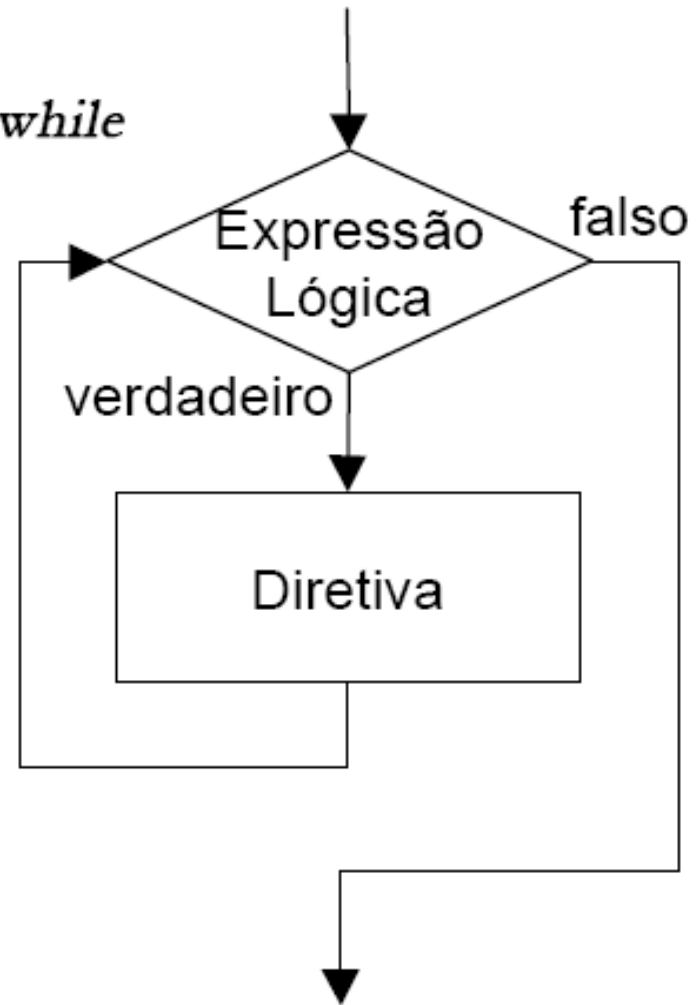
As estruturas de repetição condicionais são estruturas de repetição cujo controle de execução é feito pela avaliação de expressões condicionais. Estas estruturas são adequadas para permitir a execução repetida de um conjunto de diretivas por um número indeterminado de vezes, isto é, um número que não é conhecido durante a fase de programação mas que pode ser determinado durante a execução do programa tal como um valor a ser fornecido pelo usuário, obtido de um arquivo ou ainda de cálculos realizados com dados alimentados pelo usuário ou lido de arquivos. Existem duas estruturas de repetição condicionais: *while* e *do while*.

O *while* é o que chamamos de laço condicional, isto é, um conjunto de instruções que é repetido enquanto o resultado de uma expressão lógica (uma condição) é avaliado como verdadeiro. Abaixo segue a sintaxe desta diretiva enquanto seu o comportamento é ilustrado a seguir.

```
while (expressão_lógica)
    diretiva;
```

Note que a diretiva *while* avalia o resultado da expressão antes de executar a diretiva associada, assim é possível que diretiva nunca seja executada caso a condição seja inicialmente falsa. Um problema típico relacionado a avaliação da condição da diretiva *while* é o seguinte: se a condição nunca se tornar falsa o laço será repetido indefinidamente.

Comportamento da Diretiva *while*

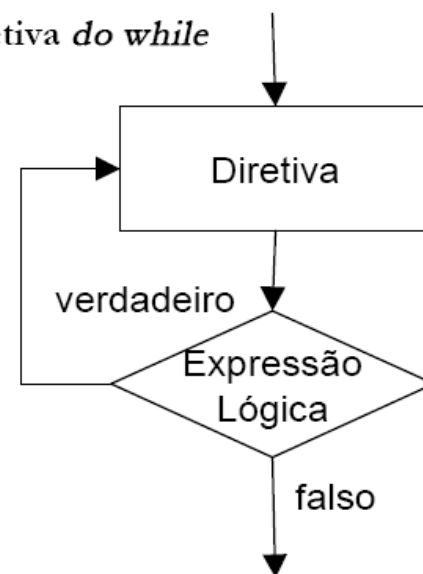


O *do while* também é um laço condicional, isto é, tal como o *while* é um conjunto de instruções repetido enquanto o resultado da condição é avaliada como verdadeira mas, diferentemente do *while*, a diretiva associada é executada antes da avaliação da expressão lógica e assim temos que esta diretiva é executada pelo menos uma vez.

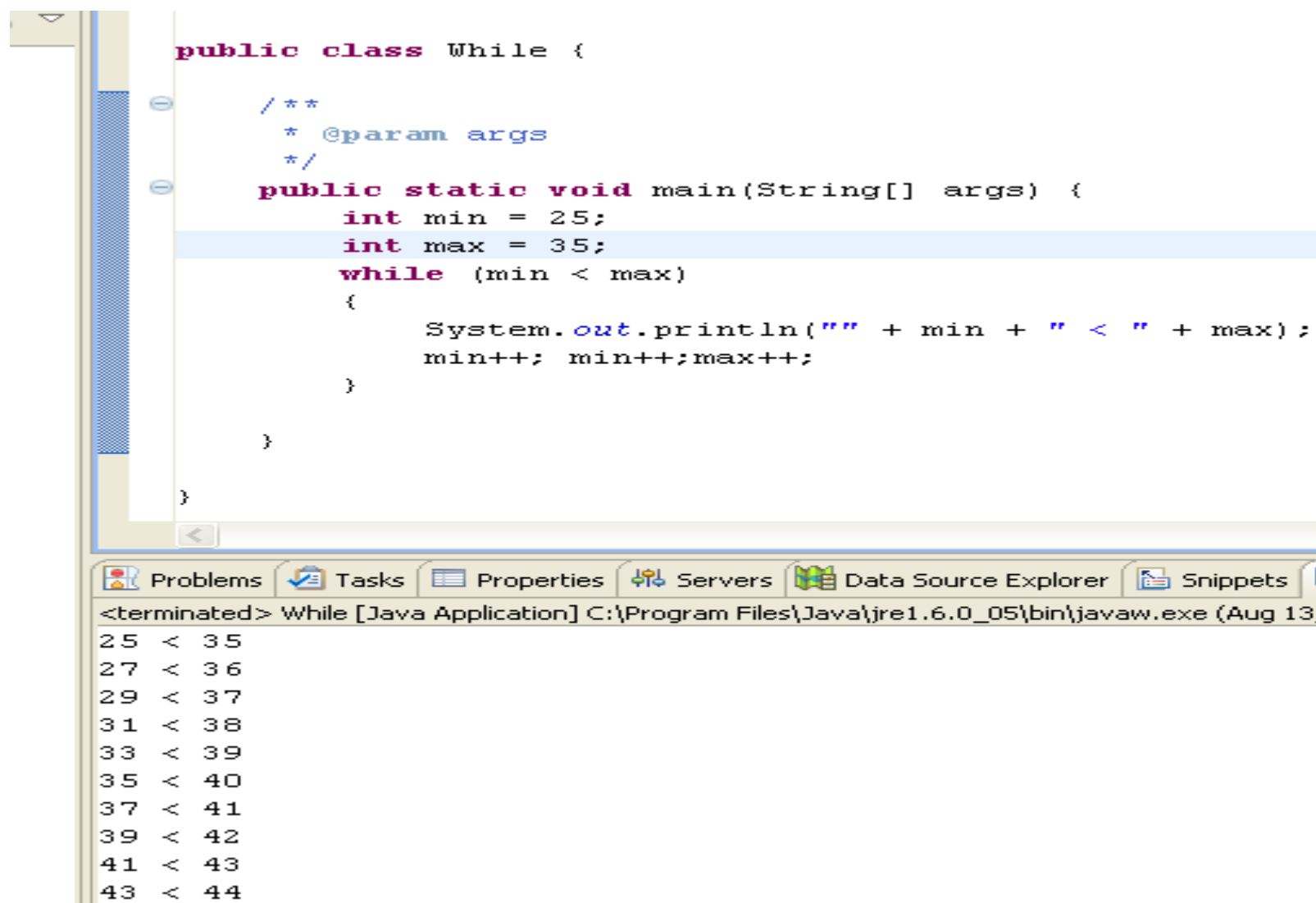
Seguem a sintaxe da diretiva *do while* e uma ilustração do seu comportamento.

```
do  
    diretiva  
while (expressão_lógica);
```

Comportamento da Diretiva *do while*



Exemplo - while



```

public class While {

    /**
     * @param args
     */
    public static void main(String[] args) {
        int min = 25;
        int max = 35;
        while (min < max)
        {
            System.out.println("'" + min + " < " + max);
            min++; min++; max++;
        }
    }
}

```

Problems Tasks Properties Servers Data Source Explorer Snippets

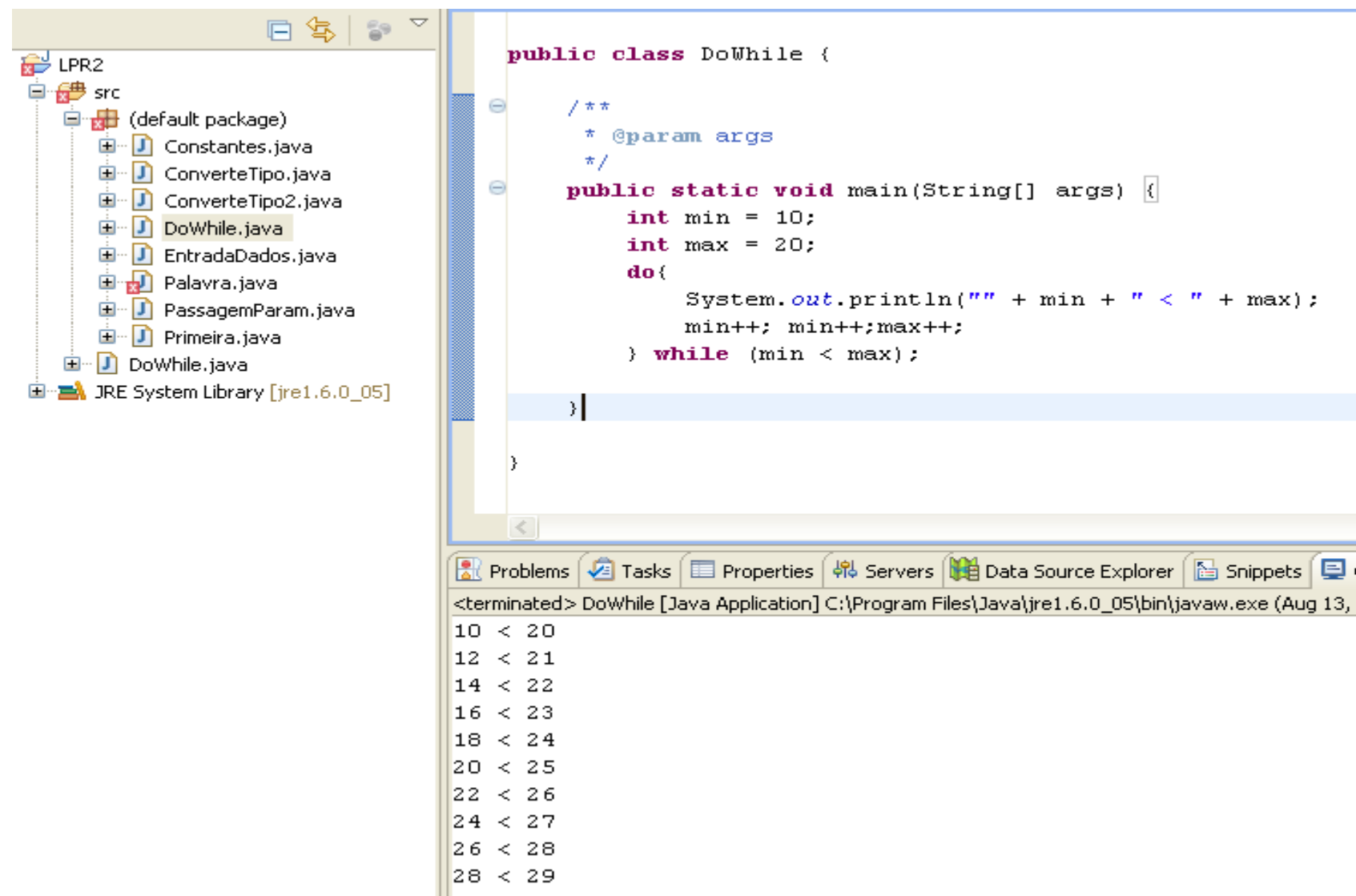
<terminated> While [Java Application] C:\Program Files\Java\jre1.6.0_05\bin\javaw.exe (Aug 13,

```

25 < 35
27 < 36
29 < 37
31 < 38
33 < 39
35 < 40
37 < 41
39 < 42
41 < 43
43 < 44

```

Exemplo – do while



The screenshot displays an IDE with a project named 'LPR2'. The 'src' folder contains a 'default package' with several Java files, including 'DoWhile.java'. The code in 'DoWhile.java' is as follows:

```
public class DoWhile {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        int min = 10;  
        int max = 20;  
        do {  
            System.out.println(" " + min + " < " + max);  
            min++; min++; max++;  
        } while (min < max);  
    }  
}
```

The output window at the bottom shows the execution results of the 'DoWhile' Java Application, displaying the following output:

```
<terminated> DoWhile [Java Application] C:\Program Files\Java\jre1.6.0_05\bin\javaw.exe (Aug 13,  
10 < 20  
12 < 21  
14 < 22  
16 < 23  
18 < 24  
20 < 25  
22 < 26  
24 < 27  
26 < 28  
28 < 29
```

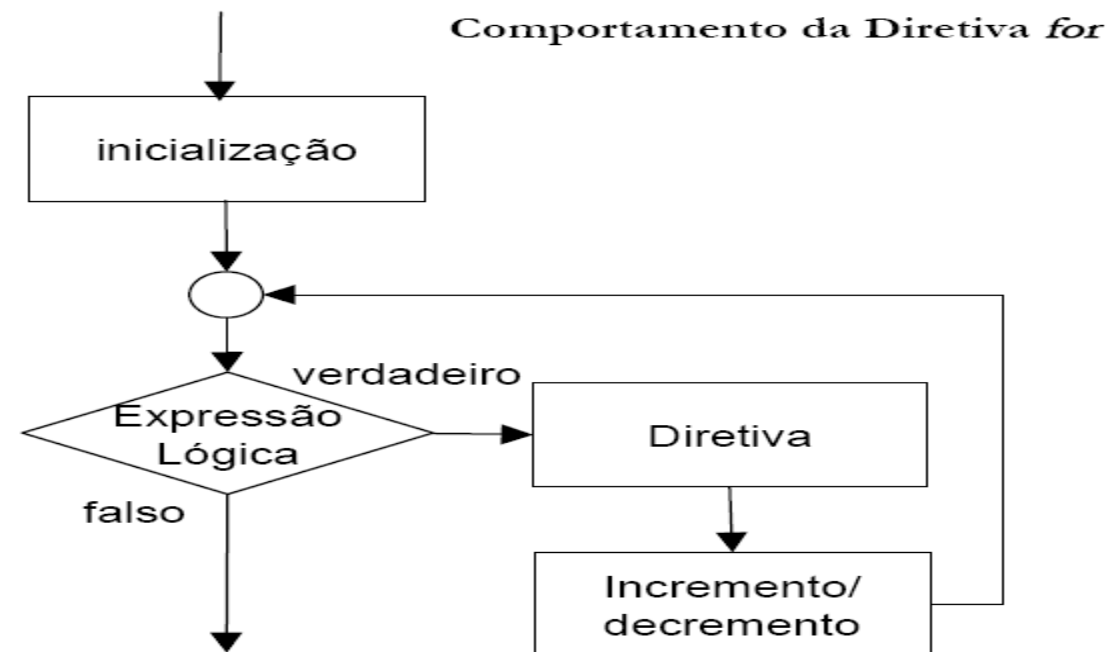
Repetição Simples - *for*

Como repetição simples consideramos um trecho de código, isto é, um conjunto de diretivas que deve ser repetido um número conhecido e fixo de vezes. A repetição é uma das tarefas mais comuns da programação utilizada para efetuarmos contagens, para obtenção de dados, para impressão etc. Em Java dispomos da diretiva *for* cuja sintaxe é dada a seguir:

```
| for (inicialização; condição de execução; incremento/decremento)  
|   diretiva;
```

O *for* possui três campos ou seções, todas opcionais, delimitados por um par de parêntesis que efetuam o controle de repetição de uma diretiva individual ou de um bloco de diretivas. Cada campo é separado do outro por um ponto e vírgula. O primeiro campo é usado para dar valor inicial a uma variável de controle (um contador). O segundo campo é uma expressão lógica que determina a execução da diretiva associada ao *for*, geralmente utilizando a variável de controle e outros valores.

Após a execução da seção de inicialização ocorre a avaliação da expressão lógica. Se a expressão é avaliada como verdadeira, a diretiva associada é executada, caso contrário o comando *for* é encerrado e a execução do programa prossegue com o próximo comando após o *for*. O terceiro campo determina como a variável de controle será modificada a cada iteração do *for*. Considera-se como iteração a execução completa da diretiva associada, fazendo que ocorra o incremento ou decremento da variável de controle.



Exemplo - for

```
import java.io.*;

public class exemploFor {
    public static void main (String args[]) {
        int j;
        for (j=0; j<10; j++) {
            System.out.println(""+j);
        }
    }
}
```

Estruturas de Erros

O Java oferece duas importantes estruturas para o controle de erros muito semelhantes as estruturas existentes na linguagem C++: *try catch* e *try finally*. Ambas tem o propósito de evitar que o programador tenha que realizar testes de verificação e avaliação antes da realização de certas operações, desviando automaticamente o fluxo de execução para rotinas de tratamento de erro. Através destas diretivas, delimita-se um trecho de código que será monitorado automaticamente pelo sistema. A ocorrência de erros no Java é sinalizada através de exceções, isto é, objetos especiais que carregam informação sobre o tipo de erro detectado. Existem várias classes de exceção adequadas para o tratamento do problemas mais comuns em Java, usualmente inclusas nos respectivos pacotes. Exceções especiais podem ser criadas em adição às existentes ampliando as possibilidades de tratamento de erro. Com o *try catch* a ocorrência de erros de um ou mais tipos dentro do trecho de código delimitado desvia a execução automaticamente para uma rotina designada para o tratamento específico deste erro.

A sintaxe do *try catch* é a seguinte:

```
try {  
    diretiva_normal;  
} catch (exception1) {  
    diretiva_de_tratamento_de_erro1;  
} catch (exception2) {  
    diretiva_de_tratamento_de_erro2;  
}
```

Exemplo

```

import javax.swing.*;
public class Exemplo_Try_Catch
{
    public static void main (String arg[])
    {
        int x = 0, y= 0, c = 0 ;

        String A = "", B = "";
        try
        {
            A = JOptionPane.showInputDialog(null, "Digite"+
            "o valor de X " );
            x = Integer.parseInt(A);

            B = JOptionPane.showInputDialog(null, "Digite"+
            "o valor de Y " );
            y = Integer.parseInt(B);

            c = x/y;
            JOptionPane.showMessageDialog(null, "A parte inteira"+
            " do resultado e "+c, "ERRO", 1 );
        }
        catch(ArithmeticException ex )
        {
            JOptionPane.showMessageDialog(null, "Voce tentou"+
            " dividir por Zero ! ", "ERRO", 1 );
        }
        catch(NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null, "O dado tem que ser"+
            " apenas numérico !", "ERRO :", 1 );
        }
        System.exit(0);
    }
}

```

Exemplo

Com o *try finally* temos um comportamento bastante diferente: uma rotina de finalização é garantidamente executada, isto é, o trecho particular de código contido na cláusula *finally* é sempre executado ocorrendo ou não erros dentro do trecho delimitado pela cláusula *try*.

A sintaxe do *try finally* é colocada a seguir:

```
try {  
    diretiva_normal;  
} finally {  
    diretiva_de_tratamento_de_erro;  
}
```

Isto é particularmente interessante quando certos recursos do sistema ou estruturas de dados devem ser liberadas, independentemente de sua utilização.

```

import javax.swing.*;
class Exemplo_Try_Catch_Finally
{
    static double compr= 0, larg= 0, prof= 0, volume ;

    public static void main(String arg[])
    {
        try
        {
            String A = JOptionPane.showInputDialog(null,"Digite o comprimento : " );
            A = A.replace(',','.');
            compr = Double.parseDouble(A);

            String B = JOptionPane.showInputDialog(null,"Digite a largura : " );
            B = B.replace(',','.');
            larg = Double.parseDouble(B);

            String C = JOptionPane.showInputDialog(null,"Digite a profundidade : " );
            C = C.replace(',','.');
            prof = Double.parseDouble(C);

        }
        catch (NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null,"Digite apenas números","Volume da Piscina",1 );
            System.exit(0);
        }
        finally
        {
            volume = (compr*larg*prof)*1000;
            volume *= 100;
            volume = Math.ceil(volume);
            volume /=100 ;

            String st = "Para a piscina solicitada "+
                "\nAs dimensões são :"+ " "+compr+" X "+
                " "+larg+" X "+prof+" ( em metros )"+
                "\n"+
                "\nO volume desta piscina = : "+volume+" litros";
            JOptionPane.showMessageDialog(null,st,"Volume da Piscina",1 );
            System.exit(0);
        }
    }
}

```

É possível que o programador crie suas exceções particulares, definidas para problemas específicos que estão sendo tratados na aplicação. Para tanto deve-se utilizar o comando ***throw***.

Em inglês esta palavra significa “lançar” ou “disparar”.

Uma nova exceção pode ser declarada fazendo-se uma chamada à instância da classe ***Exception*** com o uso do comando:

throw new Exception (“nova exceção”);

```

import javax.swing.*;
public class Exemplo_throw
{
    public static void efetuar()
    {
        int x = 12, y = 0, c = 0 ;

        try
        {
            String B = JOptionPane.showInputDialog(null, "Digite o valor de Y " );
            y = Integer.parseInt(B);

            if(x/y == 1)
            {
                throw new Exception("Nova_Exceção");
            }
            c = x/y;
            JOptionPane.showMessageDialog(null, "A parte inteira do resultado e "+c, "Resultado", 1);
        }
        catch(NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null, "Digite apenas números !", "ERRO", 1 );
            efetuar();
        }
        catch(ArithmeticException e)
        {
            JOptionPane.showMessageDialog(null, "Proibido dividir por Zero !", "ERRO", 1 );
            efetuar();
        }
        catch(Exception Nova_Exceção)
        {
            JOptionPane.showMessageDialog(null, "Valores entre 7 e 12 são proibidos !", "ERRO", 1 );
            efetuar();
        }
        System.exit(0);
    }
    public static void main (String arg[])
    {
        efetuar();
    }
}

```