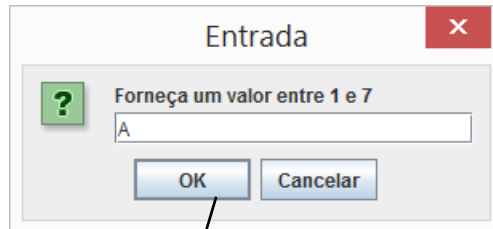


Capítulo - 08

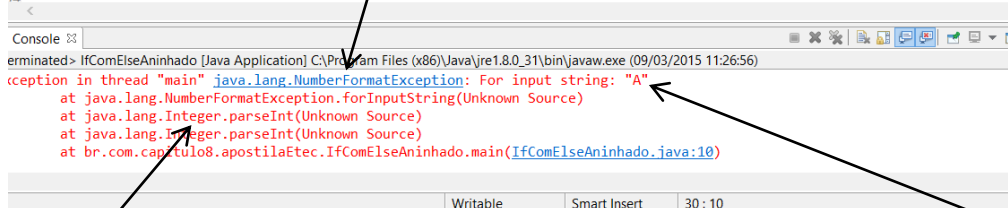
8.1 - O bloco try catch visão preliminar

As exceções em Java referem-se aos erros que podem ser gerados **durante a execução** de um programa. Como o nome sugere, trata-se de algo que interrompe a execução normal do programa. No exercício do capítulo 7 no tópico 7.6 experimente digitar uma letra como no exemplo abaixo:



Em seguida, clique em OK.

```
1 import javax.swing.JOptionPane;
2 public class IfComElseAninhado {
3
4     public static void main(String[] args) {
5         String diaDaSemana = JOptionPane.showInputDialog("Forneça um valor entre 1 e 7");
6         if (diaDaSemana != null)
7         {
8             int dia = Integer.parseInt(diaDaSemana);
9             if (dia==1)
10                 diaDaSemana = "Domingo";
11             else if (dia==2)
12                 diaDaSemana = "Segunda";
13             else if (dia==3)
14                 diaDaSemana = "Terça";
15             else if (dia==4)
16                 diaDaSemana = "Quarta";
17             else if (dia==5)
18                 diaDaSemana = "Quinta";
19             else if (dia==6)
20                 diaDaSemana = "Sexta";
21             else if (dia==7)
22                 diaDaSemana = "Sábado";
23             else
24                 diaDaSemana = "Dia da Semana desconhecido!";
25
26             JOptionPane.showMessageDialog(null, diaDaSemana);
27         }
28         else {
29             JOptionPane.showMessageDialog(null, "Cancelando sistema...");
30             System.exit(0);
31         }
32     }
33 }
34
```



Este erro ou para crítica do programa ocorreu porque o Java não conseguiu converter a letra **A** para o tipo **inteiro**, ou seja, a conversão `int dia = Integer.parseInt(diaDaSemana)` não foi possível realizar.

A este tipo de erro denominamos **erro de exceção**.

Em Java as exceções são divididas em duas categorias. Unchecked (não verificadas) e Checked (verificadas).

Existem muitas exceções que podem ser geradas pelas mais diversas classes, e enumerá-las seria algo dispendioso e desnecessário para o escopo desta apostila. Mas cada vez que uma exceção diferente for usada eu explicarei.

Vamos reconstruir a classe dos dias da semana, porém agora tratando este possível erro com o bloco **try catch**. Veja:

```

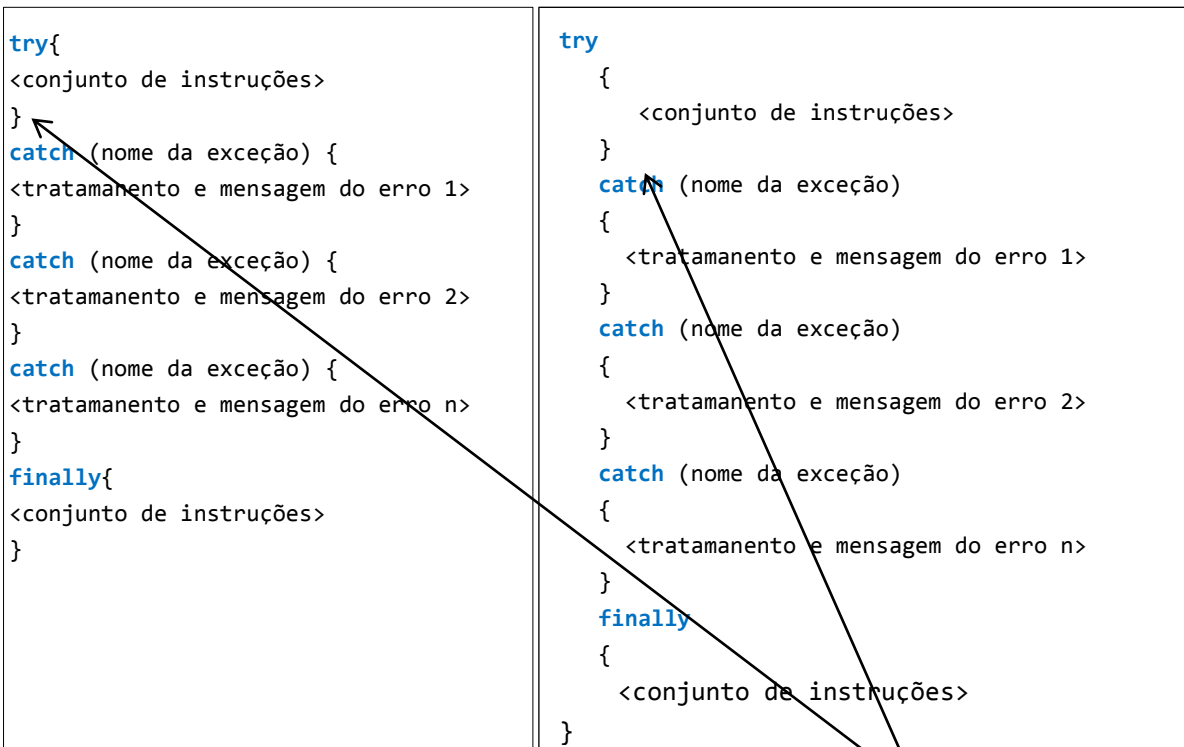
1 import javax.swing.JOptionPane;
2 public class IfComElseAninhado {
3
4     public static void main(String[] args) {
5         String diaDaSemana = JOptionPane.showInputDialog("Forneça um valor entre 1 e 7");
6         if (diaDaSemana != null) {
7             try{
8                 int dia = Integer.parseInt(diaDaSemana);
9                 if (dia==1)
10                     diaDaSemana = "Domingo";
11                 else if (dia==2)
12                     diaDaSemana = "Segunda";
13                 else if (dia==3)
14                     diaDaSemana = "Terça";
15                 else if (dia==4)
16                     diaDaSemana = "Quarta";
17                 else if (dia==5)
18                     diaDaSemana = "Quinta";
19                 else if (dia==6)
20                     diaDaSemana = "Sexta";
21                 else if (dia==7)
22                     diaDaSemana = "Sábado";
23                 else
24                     diaDaSemana = "Dia da Semana desconhecido!";
25
26                 JOptionPane.showMessageDialog(null, diaDaSemana);
27             }
28
29             catch (NumberFormatException erro){
30                 JOptionPane.showMessageDialog(null, "Diagite apenas valores numéricos - \n" +
31                     "Erro!" + erro.toString());
32             }
33         }
34         System.exit(0);
35     }
36 }

```

Tudo que estiver nesta faixa da linha 8 até a 24 o try irá tentar. Caso encontre alguma exceção ele então será tratado/pego pelo catch.

Em nosso exemplo anterior, todos os comandos contidos dentro da { } do **try** será executado normalmente até encontrar uma possível exceção. Caso encontre, automaticamente o Java passa o controle do programa ao bloco catch e executa os comandos dentro de { }.

A estrutura **try-catch** possui uma das seguintes formas abaixo, use a que desejar.



Quando um **try** é usado, obrigatoriamente em seu encerramento (imediatamente após chave final) deve existir pelo menos um **catch**, a não ser que se utilize o **finally**.

Veja o exemplo abaixo:

```
1 class TryCatchFinally
2 {
3     public void mensagens()
4     {
5         try
6         {
7             System.out.println("mensagem sem erros");
8         }
9     }
10 }
```

Ao tentar compilar a classe o compilador apresentará o seguinte erro:

```
TryCatchFinally.java:5: error: 'try' without 'catch', 'finally' or resource declarations
    try
    ^
1 error
```

Em outras palavras “você começou um try sem usar um catch. Isso não é permitido!”

Ou se você estiver utilizando o eclipse. Ele apresentará um sinal de erro na chave que finaliza o try. E, ao posicionar o cursor do mouse sobre ele, você verá a mensagem:

```
1 public class TryCatchFinally
2 {
3     public void mensagens()
4     {
5         try
6         {
7             System.out.println("mensagem sem erros");
8         }
9     }
10 }
11
```

Syntax error, insert "Finally" to complete BlockStatements
Press 'F2' for focus

Após tentar rodar no Eclipse, ele avisará!

```
1 public class TryCatchFinally
2 {
3     public static void main (String[] args)
4     {
5         try
6         {
7             System.out.println("mensagem sem erros");
8         }
9     }
10 }
11
```

Problems Javadoc Declaration Console
<terminated> TryCatchFinally [Java Application] C:\Program Files (x86)\Java\jre1.8.0_31\bin\javaw.exe (26/02/2015 17:06:14)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Syntax error, insert "Finally" to complete BlockStatements
at TryCatchFinally.main(TryCatchFinally.java:8)

O correto seria construir a classe da seguinte forma:

```
1 class Trycatchfinally
2 {
3     public void mensagens()
4     {
5         try
6         {
7             System.out.println("mensagem sem erros");
8         }
9         catch (Exception erro){
10             System.out.println("mensagem do erro" + erro.toString());
11         }
12     }
13 }
```

8.2 - Cuidado ao tratar os erros de exceção

Os erros gerados pelo programa sempre estão relacionado com as classes utilizadas.

Por exemplo:

O método `parseFloat` da classe `Float` (`Float.parseFloat`) pode gerar um erro de conversão numérica “`NumberFormatException`”. Este é o tipo de erro que pode ser tratado/previsto quando esta classe estiver em uso.

Cuidado!

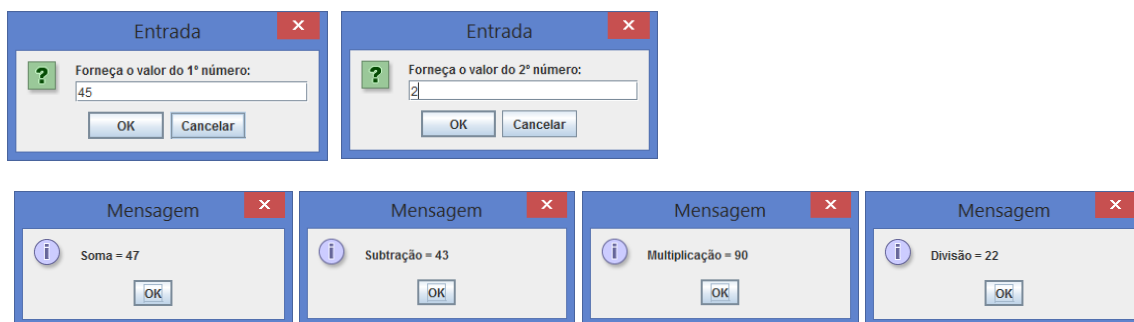
Não faz sentido tratar um erro de entrada e saída de dados (`IOException`) quando você estiver utilizando a classe `Float`. Portanto o tratamento de erros deve ser feita de forma coerente.

Mais um exemplo do uso de try-catch.

Neste exemplo o programa recebe dois números inteiros e realiza quatro operações básicas entre eles.

```
1 import javax.swing.JOptionPane;
2 public class TryCatch {
3
4     public static void main(String[] args) {
5         String aux1 = JOptionPane.showInputDialog("Forneça o valor do 1º número: ");
6         if (aux1 == null) { // if (aux1 == null || aux1.isEmpty()){
7             System.exit(0);
8         }
9
10        String aux2 = JOptionPane.showInputDialog("Forneça o valor do 2º número: ");
11        if (aux2 == null) {
12            System.exit(0);
13        }
14        try {
15            int num1 = Integer.parseInt(aux1);
16            int num2 = Integer.parseInt(aux2);
17            JOptionPane.showMessageDialog(null, "Soma = " + (num1 + num2));
18            JOptionPane.showMessageDialog(null, "Subtração = " + (num1 - num2));
19            JOptionPane.showMessageDialog(null, "Multiplicação = " + (num1 * num2));
20            JOptionPane.showMessageDialog(null, "Divisão = " + (num1 / num2));
21        }
22        catch (ArithmeticException erro) {
23            JOptionPane.showMessageDialog(null, "Erro de divisão por zero \n" + erro.toString(),
24                "Erro", JOptionPane.ERROR_MESSAGE);
25        }
26        catch (NumberFormatException erro) {
27            JOptionPane.showMessageDialog(null, "Digite apenas números inteiros \n" + erro.toString(),
28                "Erro", JOptionPane.ERROR_MESSAGE);
29        }
30    }
31 }
```

Neste exemplo podemos entender as funcionalidades do try-catch.



Nas linhas de 5 a 8 o programa encarrega-se da solicitação e armazenamento do valor. Caso o usuário clique em cancelar o programa encerrará.

Nas linha de 10 a 13 a operação acima será repetida.

Agora rode novamente e coloque 45 para o 1º número solicitado e 0 para o segundo e veja o resultado.

Tente novamente com os valores 1.3 na primeira pergunta e ‘ na segunda e veja o resultado.

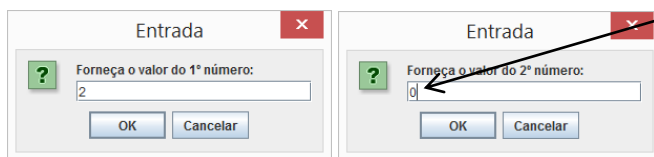
8.3 - Novidades no tratamento de erros múltiplos catch

A linguagem Java tem uma novidade inserida a partir da versão 7 que é a possibilidade de tratar mais de uma exceção em um mesmo `catch`. Com isso as linha de 22 a 29 do nosso exemplo anterior poderiam ser tratadas no mesmo `catch`.

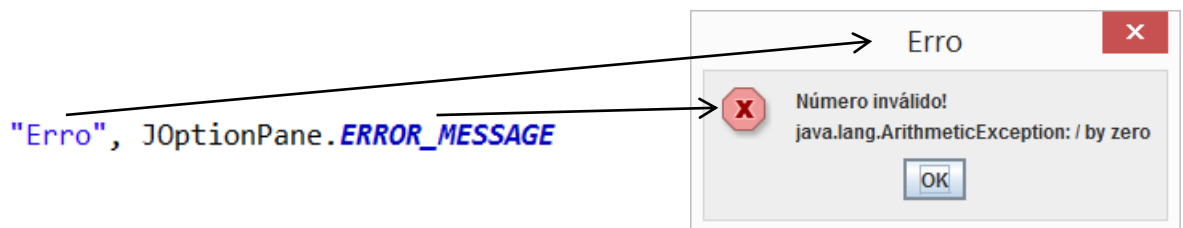
Veja:

```
23     catch (ArithmeticException | NumberFormatException | NullPointerException erro){
24         JOptionPane.showMessageDialog(null, "Número inválido! \n"+ erro.toString(),
25             "Erro", JOptionPane.ERROR_MESSAGE);
26     }
```

O catch acima será capaz de capturar, por exemplo, erro de divisão por zero quando se insere estes valores:



Gerando a mensagem:



Então altere seu código para:

```
1 import javax.swing.JOptionPane;
2
3 public class TryCatchMultiplo {
4
5     public static void main(String[] args) {
6         String aux1 = JOptionPane.showInputDialog("Forneça o valor do 1º número: ");
7         if (aux1 == null){
8             System.exit(0);
9         }
10
11         String aux2 = JOptionPane.showInputDialog("Forneça o valor do 2º número: ");
12         if (aux2 == null){
13             System.exit(0);
14         }
15         try{
16             int num1 = Integer.parseInt(aux1);
17             int num2 = Integer.parseInt(aux2);
18             JOptionPane.showMessageDialog(null, "Soma = " + (num1 + num2));
19             JOptionPane.showMessageDialog(null, "Subtração = " + (num1 - num2));
20             JOptionPane.showMessageDialog(null, "Multiplicação = " + (num1 * num2));
21             float resp = num1 / num2;
22             JOptionPane.showMessageDialog(null, "Divisão = " + resp);
23         }
24         catch (ArithmeticException | NumberFormatException | NullPointerException erro){
25             JOptionPane.showMessageDialog(null, "Número inválido! \n"+ erro.toString(),
26                 "Erro", JOptionPane.ERROR_MESSAGE);
27         }
28     }
29 }
```

8.4 – O bloco *finally*

O objetivo do **finally** é manter códigos para a liberação de recursos, adquiridos em seu bloco **try** correspondente. Um bom exemplo para isto é a abertura de um banco de dados para uma consulta, alteração ou inserção de valores. Após ter completado a execução passando pelo **try** o **finally** poderá, por exemplo, fechar a conexão. Ou ainda se o **try** falhar e passar pelo **catch** obrigatoriamente o **finally** também fechará o banco de dados liberando recursos.

Outra explicação caso não tenha entendido!

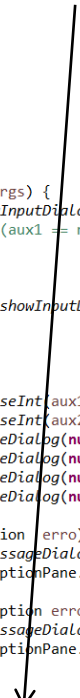
Durante a execução normal de um programa, caso nenhuma exceção ocorra, os blocos **catch** não são executados e o controle prossegue para o bloco **finally**, que libera o recurso e passa a executar a primeira instrução após o bloco **finally**. Caso ocorra alguma exceção, a execução do programa é desviada para o grupo de instruções após o bloco **finally**. Ou seja, passando ou não pelo **catch** o **finally** é executado após o **try**!

Para que você possa verificar o funcionamento do **finally**, modifique o trecho final do programa acima, inserindo o código:

```
finally {  
    JOptionPane.showMessageDialog(null, "Fim da execução!");  
}
```

Veja:

```
2 import javax.swing.JOptionPane;  
3 public class TryCatchFinally {  
4  
5     public static void main(String[] args) {  
6         String aux1 = JOptionPane.showInputDialog("Forneça o valor do 1º número: ");  
7         if (aux1 == null) { // if (aux1 == null || aux1.isEmpty()) {  
8             System.exit(0);  
9         }  
10  
11         String aux2 = JOptionPane.showInputDialog("Forneça o valor do 2º número: ");  
12         if (aux2 == null) {  
13             System.exit(0);  
14         }  
15         try {  
16             int num1 = Integer.parseInt(aux1);  
17             int num2 = Integer.parseInt(aux2);  
18             JOptionPane.showMessageDialog(null, "Soma = " + (num1 + num2));  
19             JOptionPane.showMessageDialog(null, "Subtração = " + (num1 - num2));  
20             JOptionPane.showMessageDialog(null, "Multiplicação = " + (num1 * num2));  
21             JOptionPane.showMessageDialog(null, "Divisão = " + (num1 / num2));  
22         }  
23         catch (ArithmeticException erro) {  
24             JOptionPane.showMessageDialog(null, "Erro de divisão por zero \n" + erro.toString(),  
25                 "Erro", JOptionPane.ERROR_MESSAGE);  
26         }  
27         catch (NumberFormatException erro) {  
28             JOptionPane.showMessageDialog(null, "Digite apenas números inteiros \n" + erro.toString(),  
29                 "Erro", JOptionPane.ERROR_MESSAGE);  
30         }  
31  
32         finally {  
33             JOptionPane.showMessageDialog(null, "Fim da execução!");  
34         }  
35     }  
36 }
```



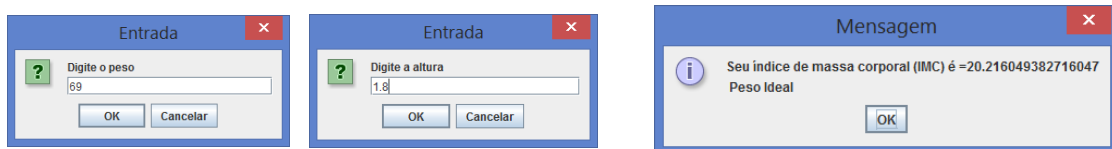
Ao executar o exemplo, agora com o uso do **finally**, a linha 33 sempre será executada isto é, sempre será emitida a mensagem “Fim da execução” independente de existirem erros ou não durante o processo de execução.

Atividade-08

1) Usando a classe **JOptionPane** para a entrada de dados, elabore uma classe que realize o cálculo do peso ideal. O peso ideal é dado pelo IMC (índice de massa corporal). Considere a seguinte tabela:

Índice IMC	Descrição
Menor que 18,5	Peso abaixo do normal
Entre 18,5 e 24,4	Peso ideal
Entre 24,5 e 29,9	Pré-obesidade
Entre 30 e 34,9	Obesidade classe I
Entre 35 e 39,9	Obesidade classe II (severa)
Maior que 39,9	Obesidade classe II (mórbida)

Obs. Para calcular o IMC use a fórmula **IMC = peso / altura²**. Solicite o peso e a altura do usuário, faça o cálculo e represente a mensagem correspondente. Veja o resultado da execução na Figura abaixo.



2) Elabore uma classe para cálculo do salário líquido em que três valores devem ser informados pelo usuário. A quantidade de horas trabalhadas, salário por hora e o número de dependentes. O programa deve mostrar na tela as informações que estão no lado esquerdo da tabela seguinte. Os cálculos correspondentes aparecem no lado direito.

INFORMAÇÃO	CÁLCULO
Salário Bruto	Horas trabalhadas * valor da Hora +(50 * nº Dependentes)
Desconto INSS	Se o salário Bruto <= 1.247,70 = Salário Bruto * 8/100 De 1.247,71 até 2.079,50 = Salário Bruto * 9/100 De 2.079,51 até 4.159,00 = Salário Bruto * 11/100 Acima de 4.159,00 descontar 457,28 = Salário Bruto – 457,28
Desconto IR	Se o Salário Bruto < 700 IR =0 Se o Salário Bruto > 700 até 1000 IR = Salário Bruto * 5/100 Se o Salário Bruto > 1000 IR = Salário Bruto * 7/100
Salário Líquido	Salário Bruto – INSS - IR

As imagens abaixo ilustra como os resultados devem aparecer na tela ao executar a classe:

