

Qualificação Profissional de Assistente de
Desenvolvimento de Sistemas

LÓGICA DE PROGRAMAÇÃO

GEEaD - Grupo de Estudos de Educação a Distância Centro de Educação Tecnológica Paula Souza São Paulo – SP, 2019

Expediente

PROGRAMA NOVOTEC VIRTUAL
GOVERNO DO ESTADO DE SÃO PAULO
EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO
QUALIFICAÇÃO PROFISSIONAL DE ASSISTENTE DE DESENVOLVIMENTO DE SISTEMAS
LÓGICA DE PROGRAMAÇÃO

PÚBLICO ALVO: ALUNOS DA 3ª SÉRIE DO ENSINO MÉDIO
TEMPO DE INTEGRALIZAÇÃO: 34 SEMANAS

Autores:

*Eliana Cristina Nogueira Barion
Marcelo Fernando Iguchi
Paulo Henrique Mendes Carvalho
Rute Akie Utida*

Revisão Técnica:

Sandra Maria Leandro

Revisão Gramatical:

Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação:

Flávio Biazim

AGENDA 4

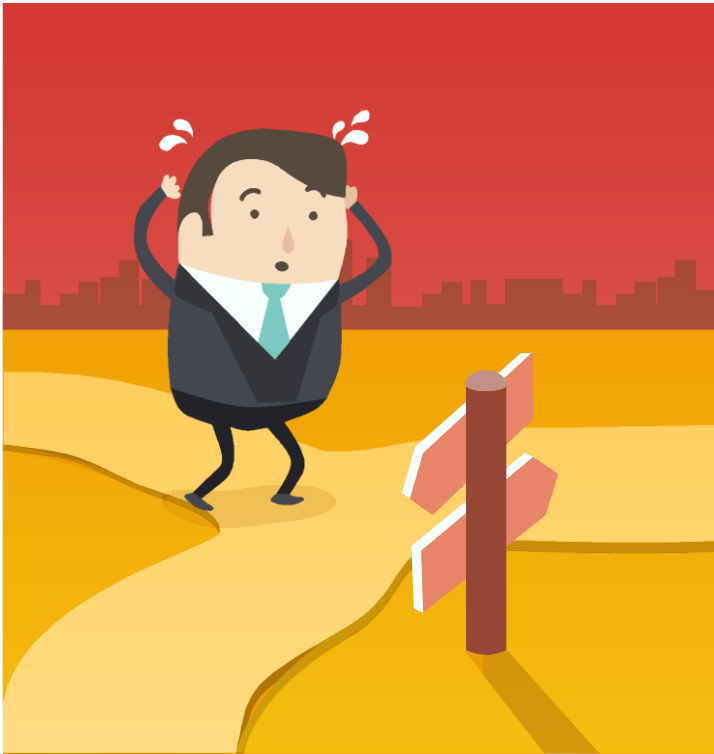
ESTRUTURAS DE DECISÃO





MERGULHANDO NO TEMA...

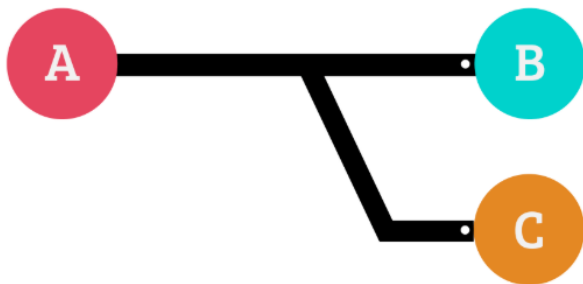
Estruturas de Decisão



Até agora trabalhamos somente com programas que realizavam tarefas simples como a realização de entrada e saída de dados e pequenos cálculos matemáticos. Contudo, os algoritmos criados até aqui não possuem poder de decisão. Ou seja, eles sempre executam as mesmas tarefas independentemente dos resultados obtidos. Infelizmente, na vida real, temos que tomar decisões que muitas vezes são difíceis e que podem alterar o rumo de nossas vidas. Com os programas ocorre a mesma coisa.

Em programação, chamamos essas decisões de Estrutura de decisão. No Java, chamamos de comando IF.

Imagem 01: Freepik – Tomada de Decisão: Homem indeciso sobre qual caminho seguir



Estruturas de decisão IF (Se...Fim)

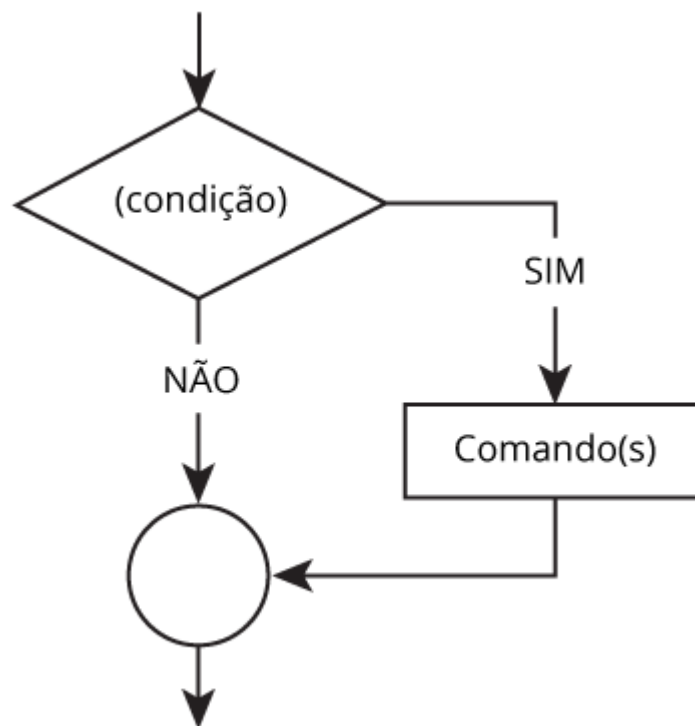
As estruturas de decisão ou testes condicionais nos permite executar um conjunto diferente de comandos dependendo do resultado de um teste utilizando operadores relacionais. Este resultado pode ser verdadeiro ou falso conforme podemos visualizar na tabela a seguir

PSEUDOCÓDIGO

FLUXOGRAMA

JAVA

SE (condição) **Então**
 {comando(s)}
Fim-Se



```

if (condição){
  {comando(s)};
}
  
```

Exemplificando...



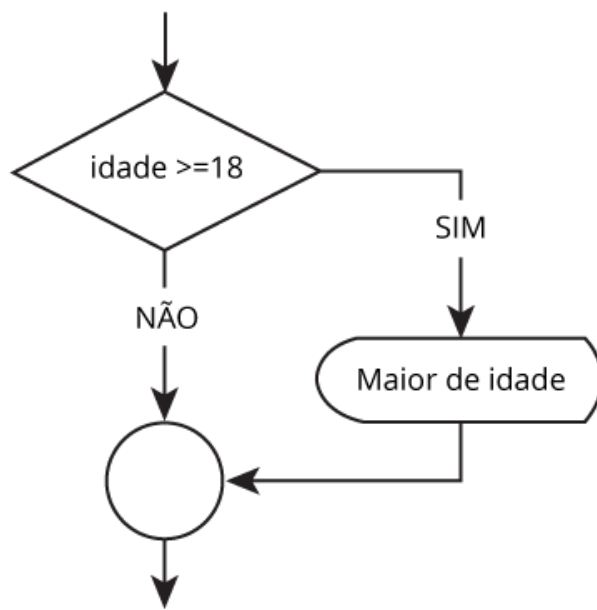
Imagine que em um determinado trecho de um programa precisamos tomar uma decisão para saber se uma pessoa é maior de idade. O programa codificado é apresentado a seguir:

PSEUDOCÓDIGO

FLUXOGRAMA

JAVA

SE (idade >=18)
Então
 Escreva
 ("Maior de
 idade")
Fim-Se



```

if (idade >=18){

OptionPane.showMessageDialog(null,
"Maior de idade");
}
  
```

O comando condicional **SE** testa a condição **idade >=18**, ou seja, se a idade for maior ou igual a 18 anos **Então** condição sim ou verdadeira será executado o comando **Escreva ("Maior de idade")**, caso contrário nada será feito (**Fim-Se**). Se observarmos o fluxograma fica mais fácil de compreender o que ocorre.

Na programação Java, conforme a sintaxe apresentada e tomando como base o exemplo da tabela acima, a palavra **SE** é substituída pelo **IF**, a comparação permanece a mesma, a palavra **Então** é substituída pela **{**, o comando Escreva é substituído pelo **OptionPane.showMessageDialog** e finalmente o **Fim-Se** é substituído pelo **}**.

Realizando a codificação em Java de um programa completo teremos:

```

import javax.swing.JOptionPane;

public class IfSimples {

    public static void main(String[] args) {
        //declaração de variáveis
        int idade; // armazena a idade
        String aux; //variável auxiliar

        //entrada de dados
        aux = JOptionPane.showInputDialog("Entre com a sua idade");
        //conversão de tipos
        idade = Integer.parseInt(aux);

        //Decisão
        if (idade >=18) {
            JOptionPane.showMessageDialog(null, "Maior de Idade");
        } //fim do if
    } //fim do main

} //fim da classe
  
```

Explicando a Estrutura de decisão presente na linha 16, o comando de decisão if (idade >=18) irá executar o que estiver dentro das chaves até a linha 18, se e somente se o valor da idade for maior ou igual a 18. Caso contrário, não executará nenhum comando adicional e o programa será encerrado.

Se, ao executarmos o programa, digitarmos a idade de 18 anos, por exemplo:

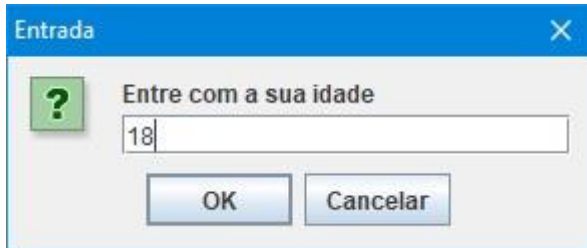


Imagem 02: Caixa de Entrada de Dados do tipo “Question” com a mensagem “Entre com a sua idade”. Na caixa de entrada de dados foi digitado o número 18. À esquerda tem o botão “OK” e à direita o botão “Cancelar”.

Teremos o seguinte retorno:

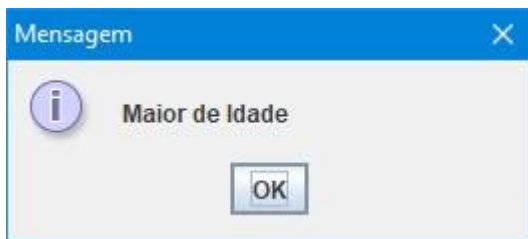
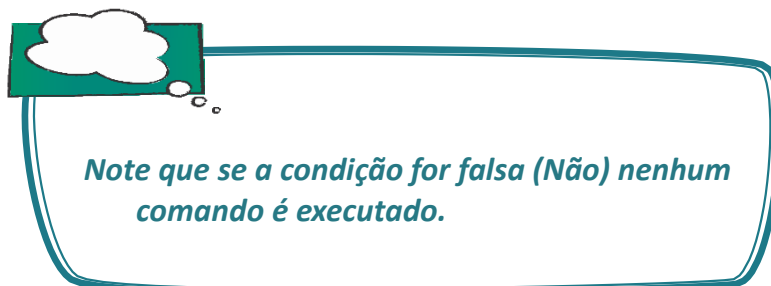


Imagem 03: Caixa de Saída de Dados informando que a idade digitada é “Maior Idade” e um botão de “OK”.



Estrutura de decisão IF-else (se...Senão...Fim se)

Acabamos de ver uma estrutura de decisão que somente realiza uma ação distinta caso o teste condicional seja verdadeiro. Contudo, em geral, também necessitamos que alguma ação seja tomada caso o teste condicional seja falso. Para isso, temos o comando If-Else:

PSEUDOCÓDIGO	FLUXOGRAMA	JAVA
SE (condição) Então {comando(s) condição verdadeira} Senão {Comando(s) condição falsa} Fim-Se	<pre> graph TD Start(()) --> Cond{condição} Cond -- NÃO --> False[Comando(s) Condição Falsa] Cond -- SIM --> True[Comando(s) Condição Verdadeira] False --> Join(()) True --> Join Join --> End(()) </pre>	<pre> if (condição){ {comando(s) condição verdadeira}; } else { {comando(s) condição falsa}; } </pre>

Imagem 04: GEEaD – Representação de Pseudocódigo, Fluxograma e Codificação Java – Sintaxe do Comando “Se...Senão”

Observando a tabela acima, você pode notar que, caso o teste lógico condicional falhe (ou seja, caso a condição não seja atendida), temos um comando ou grupo de comandos a serem executados: os comandos indicados após o **Senão**.

Retomando o exemplo anterior...



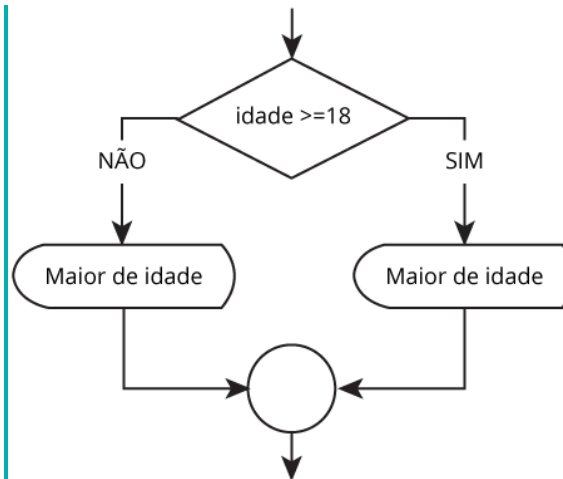
Considere o programa que analisava se uma pessoa era ou não maior de idade. Agora, porém, ele conta com comandos que serão executados se o teste condicional for falso.

PSEUDOCÓDIGO

FLUXOGRAMA

JAVA

SE (idade >=18) **Então**
 Escreva ("Maior de idade")
Senão
 Escreva ("Menor de idade")
Fim-Se



```

if (idade >= 18){
    JOptionPane.showMessageDialog
      (null, "Maior de
idade");
}
else {
    JOptionPane.showMessageDialog
      (null, "Menor de
idade");
}
  
```

Imagem 05: GEEaD – Representação de Pseudocódigo, Fluxograma e Codificação Java – Comando “Se” para verificar se a idade é maior que 18 anos.

Percebemos que é praticamente idêntico ao exemplo anterior, porém caso o teste condicional falhe (resultado falso – não) executamos um comando que exibe a mensagem “menor de idade” para o usuário. Isso é feito por meio da utilização da cláusula Senão no Pseudocódigo e else no Java.

Programa completo codificado em Java:

```

import javax.swing.JOptionPane;

public class ifComposto {

    public static void main(String[] args) {
        //declaração de variáveis
        int idade; // armazena a idade
        String aux; //variável auxiliar

        //entrada de dados
        aux = JOptionPane.showInputDialog("Entre com a sua idade");
        //conversão de tipos
        idade = Integer.parseInt(aux);

        //Decisão
        if (idade >= 18) {
            //comandos se a condição for verdadeira
            JOptionPane.showMessageDialog(null, "Maior de Idade");
        } else {
            //comandos se a condição for falsa
            JOptionPane.showMessageDialog(null, "Menor de Idade");
        } // fim do if
    } //fim do main

} //fim da classe
  
```

Se digitarmos 17 na caixa de diálogo, o resultado será:

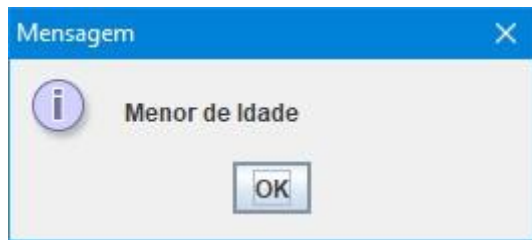
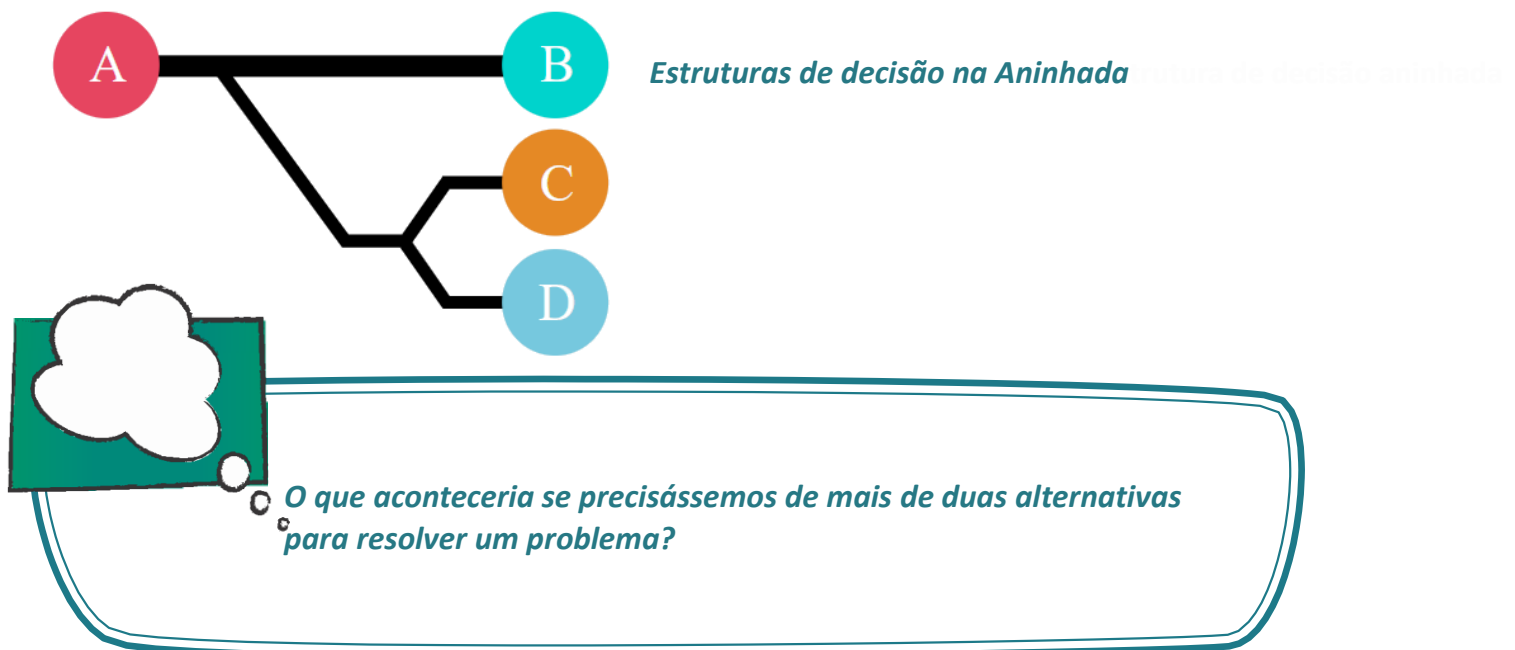


Imagem 05: Caixa de Mensagem com a informação “Menor idade” e um botão de “OK” centralizado na tela.



Talvez você tenha reparado que em uma Estrutura de Decisão podemos ter somente duas saídas: verdadeiro ou falso. Mas o que ocorre quando necessitamos de uma saída com mais de duas alternativas simultaneamente? Essa situação é bastante comum e, para isso, usamos as **Estruturas de Decisão Aninhadas**, que consistem em utilizar um comando **SE** encadeado no interior de outro.

Retomando o exemplo anterior...

Pensando no software que analisa a maioria de um indivíduo, suponha que você também queira verificar se a idade é igual a 18 anos. Veja como ficaria a codificação do programa:

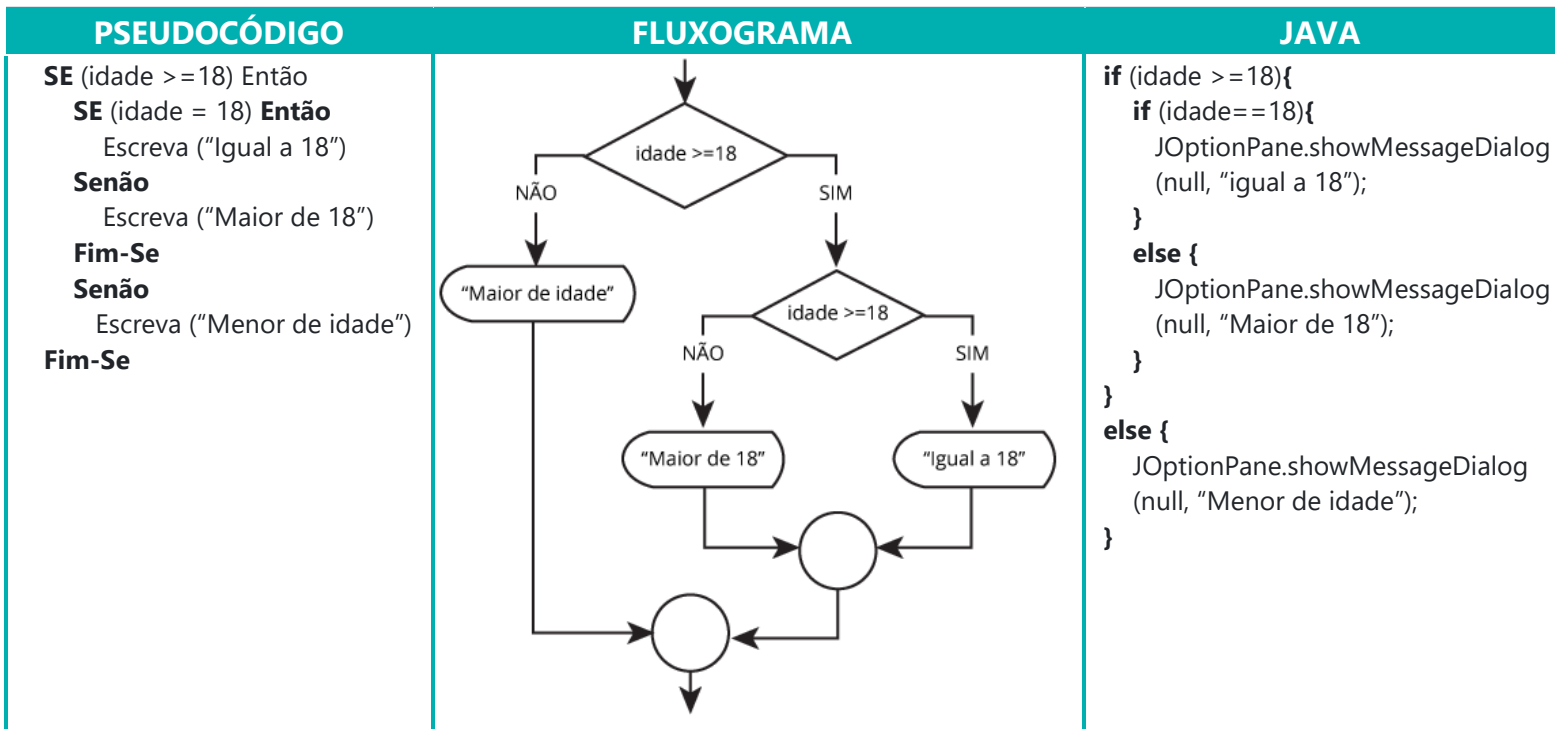


Imagem 11: GEEaD – Representação de Pseudocódigo e Codificação Java – Comando “Se..Senão” para verificar se a idade é maior que 18 anos.

Note que ao executar a primeira tomada de decisão se (idade>=18) em caso verdadeiro sabe-se somente que a idade é maior ou igual a 18. Para saber se a idade é igual a 18, é necessária a execução de outra estrutura de decisão se (idade=18). Em caso afirmati- vo sabemos que é igual e em caso negativo sabemos que é maior de 18 anos (maior de 18). É isso que chamamos de estrutura de decisão aninhada.

Veja o código em Java:

```

import javax.swing.JOptionPane;

public class ifAninhado {

    public static void main(String[] args) {
        //declaração de variáveis
        int idade; // armazena a idade
        String aux; //variável auxiliar

        //entrada de dados
        aux = JOptionPane.showInputDialog("Entre com a sua idade");
        //conversão de tipos
        idade = Integer.parseInt(aux);

        //Decisão
        if (idade >=18) { // primeiro if
            //comandos se a condição for verdadeira
            if (idade == 18) { // segundo if
                JOptionPane.showMessageDialog(null, "igual a 18");
            }else {
                JOptionPane.showMessageDialog(null, "Maior de 18");
            }//fim do segundo if
        } else {
            //comandos se a condição for falsa
            JOptionPane.showMessageDialog(null, "Menor de Idade");
        } // fim do primeiro if
    } //fim do main
} //fim da classe
          
```

Conhecendo um novo exemplo:



Agora, veja esse segundo exemplo: suponha que você precise fazer um programa em que o usuário insira um número de 1 a 7 e o programa apresente qual é o dia da semana correspondente. Você sabe que domingo é o início da semana, correspondendo ao número 1 e assim sucessivamente.

Como você poderia resolver esse programa?

1. Primeiramente, você deve pensar nas variáveis necessárias:

Como o usuário deverá inserir um número de 1 a 7, é preciso que exista uma variável que vamos chamar de **entrada**.

Qual seria o **tipo da variável** entrada?

Se você pensou “inteiro”, acertou, pois a variável irá armazenar somente números inteiros.

Será que mais alguma variável é necessária?

A resposta é não, porque vamos fazer a saída diretamente imprimindo na tela.

2. Em seguida, pense no fluxograma, pois é mais fácil de entender e visualizar:

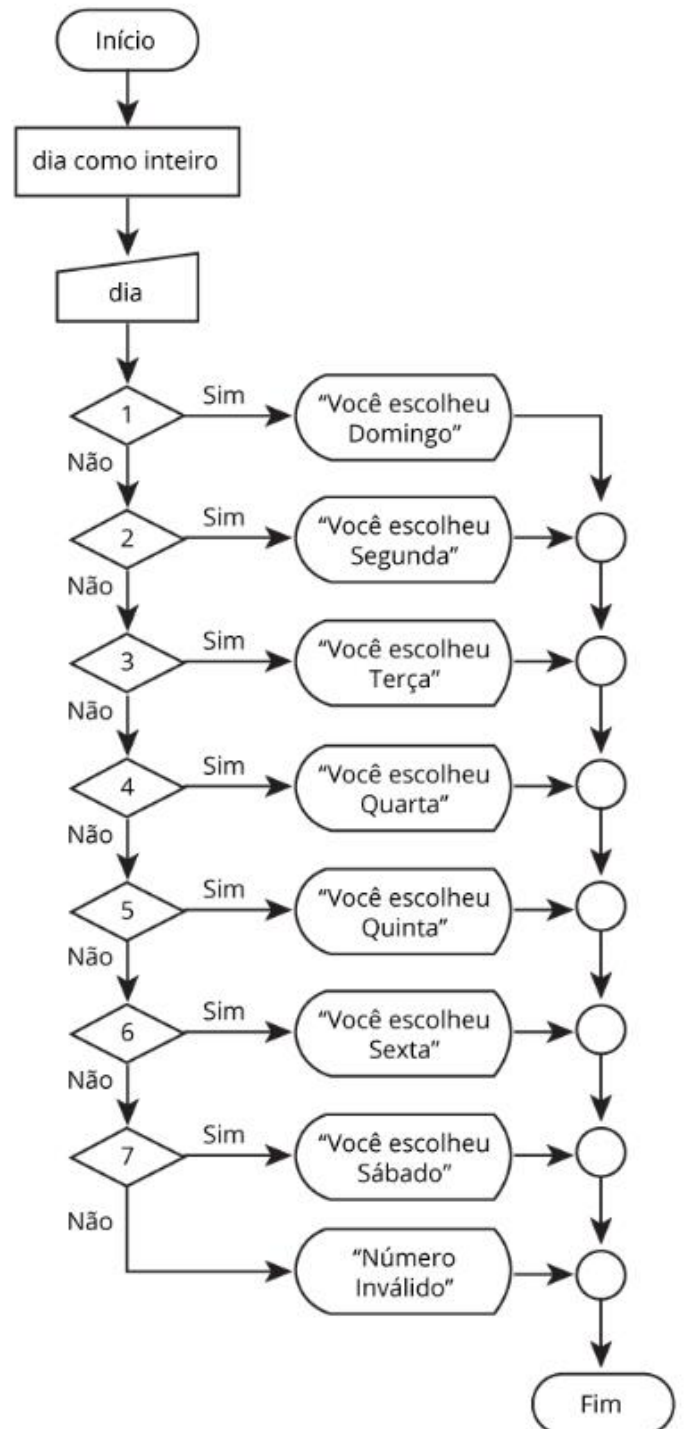
Você pode também optar por escrever o pseudocódigo. Escolha entre o fluxograma ou pseudocódigo, aquele que você preferir para simbolizar a sequência lógica do seu programa.

PSEUDOCÓDIGO

```

Programa Semana
Declare
    dia como inteiro
Início
    Escreva("Digite um Número de 1 a 7")
    Leia(dia)
    Se (dia = 1) Então
        Escreva ("Você escolheu domingo")
    senão
        Se (dia = 2) Então
            Escreva ("Você escolheu segunda")
        senão
            Se (dia = 3) Então
                Escreva ("Você escolheu terça")
            senão
                Se (dia = 4) Então
                    Escreva ("Você escolheu quarta")
                senão
                    Se (dia = 5) Então
                        Escreva ("Você escolheu quinta")
                    senão
                        Se (dia = 6) Então
                            Escreva ("Você escolheu sexta")
                        senão
                            Se (dia = 7) Então
                                Escreva ("Você escolheu sábado")
                            senão
                                Escreva("Número Inválido")
                            Fim-Se
                        Fim-Se
                    Fim-Se
                Fim-Se
            Fim-Se
        Fim-Se
    Fim-Se
Fim.
  
```

FLUXOGRAMA



3. Agora que você já pensou no problema na sua linguagem, fica mais fácil de traduzir para a linguagem Java. Veja:

```
import javax.swing.JOptionPane;

public class ifAninhadoSemana {

    public static void main(String[] args) {
        //declaração de variáveis
        int dia; // variável para armazenamento da semana

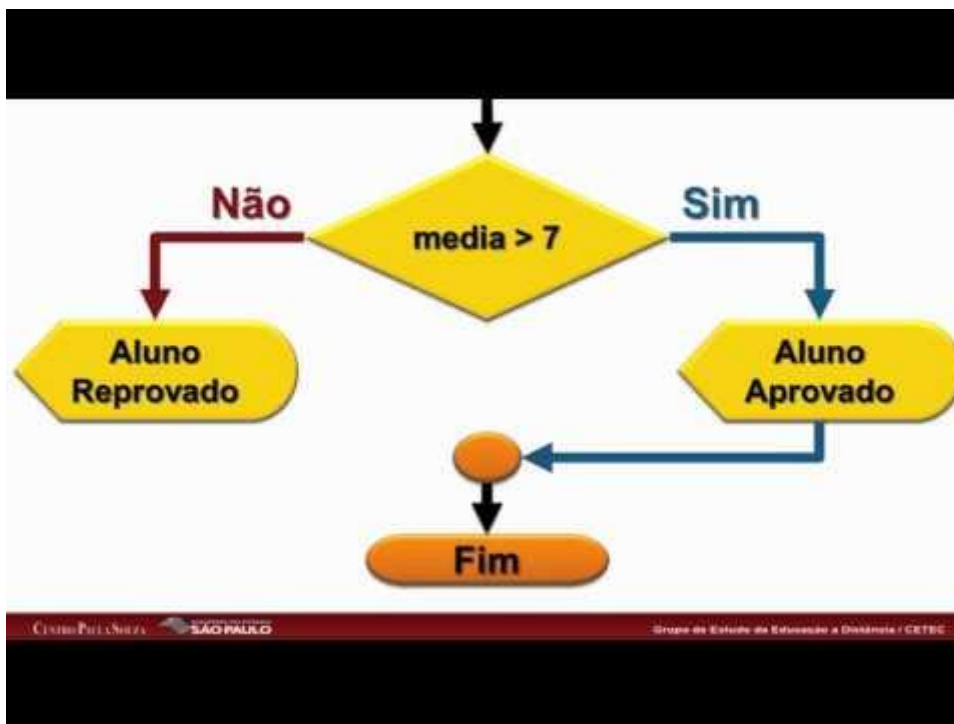
        //entrada de dados com conversão de tipos juntas
        dia = Integer.parseInt(JOptionPane.showInputDialog("Entre com um
número de 1 a 7"));

        //processamento

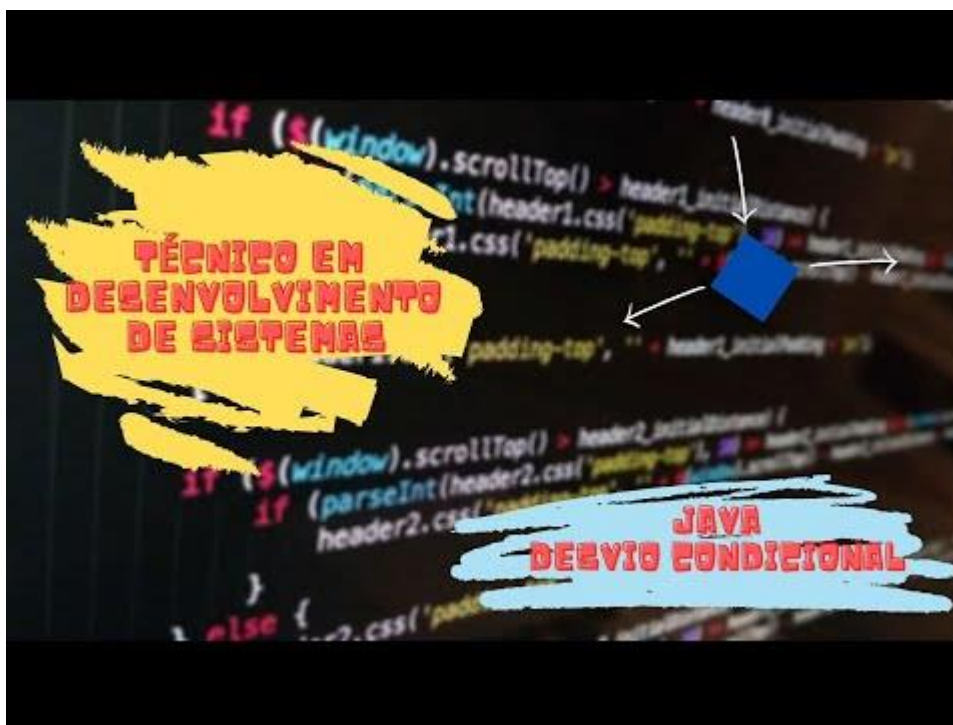
        if (dia == 1) { //if 1
            JOptionPane.showMessageDialog(null, "Você escolheu Domingo");
        } else {
            if (dia == 2) { //if 2
                JOptionPane.showMessageDialog(null, "Você escolheu
Segunda");
            } else {
                if (dia == 3) { //if 3
                    JOptionPane.showMessageDialog(null, "Você escolheu
Terça");
                } else {
                    if (dia == 4) { //if 4
                        JOptionPane.showMessageDialog(null, "Você
escolheu Quarta");
                    } else {
                        if (dia == 5) { //if 5
                            JOptionPane.showMessageDialog(null, "Você
escolheu Quinta");
                        } else {
                            if (dia == 6) { //if 6
                                JOptionPane.showMessageDialog(null,
"Você escolheu Sexta");
                            } else {
                                if (dia == 7) { //if 7
                                    JOptionPane.showMessageDialog(null, "Você escolheu Sábado");
                                } else {
                                    JOptionPane.showMessageDialog(null, "Número Inválido");
                                } // fim do if 7
                            } // fim do if 6
                        } // fim do if 5
                    } // fim do if 4
                } // fim do if 3
            } // fim do if 2
        } // fim do if 1
    } // fim do método main
} // fim da classe
```

Vamos assistir aos vídeos do Prof. Sandro Valérius para complementar nossos estudos:

Lógica – Desvio Condicional SE:



Lógica e Programação em Java – Desvio Condicional SE/if:



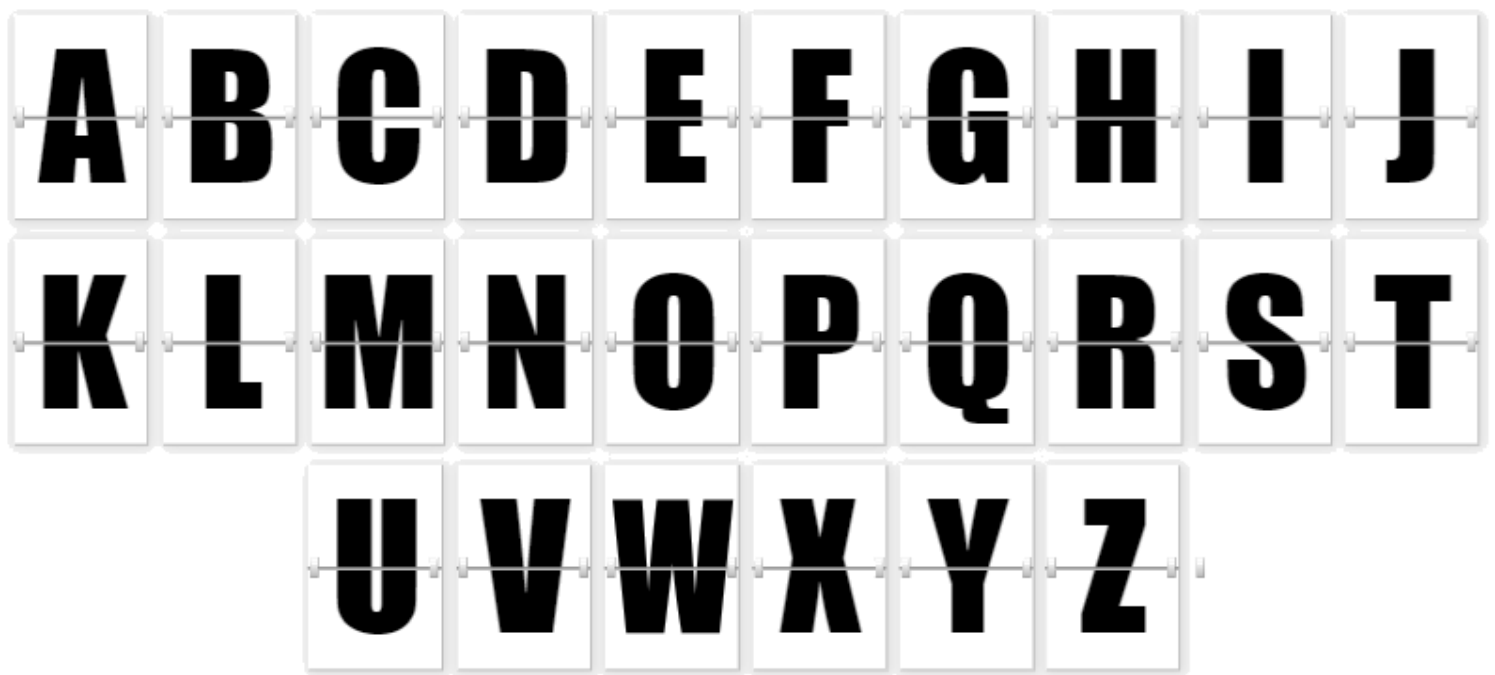


VOCÊ NO COMANDO

Observando os exemplos apresentados sobre estruturas de decisão aninhadas, existe alguma relação entre o número de alternativas e o número necessário de comparações a serem efetuadas? Reflita sobre o assunto.

Se você acredita que sim, você acertou! Existe sim uma relação. O número de comparações necessárias em um programa é o número de alternativas menos uma unidade. Ou seja, se temos 6 alternativas, teremos 5 comparações em nosso programa.

Como realizar comparações com String no Java



Já pensou como poderíamos realizar comparações com uma sequência de caracteres em Java? Seria a mesma forma que utilizamos com tipos numéricos?

Para realizarmos uma comparação de um conteúdo de uma variável com uma String – sequência de caracteres – no Java temos que utilizar um método especial o **.equals()**.

Mas como utilizamos o **.equals()**? Vejamos o exemplo a seguir:

```

import javax.swing.JOptionPane;

public class IfEquals {

    public static void main(String[] args) {
        //declaração de variáveis
        String nome;

        //entrada de dados
        nome = JOptionPane.showInputDialog("Entre com um nome");

        //Processamento e saída
        if (nome.equals("Jose")) {
            JOptionPane.showMessageDialog(null, "O Nome Digitado é Jose");
        } else {
            JOptionPane.showMessageDialog(null, "O nome digitado foi " +
nome);
        }
    }
}

```

Note que na linha 13 utilizamos a expressão **nome.equals("Jose")**, ou seja, estamos comparando se o valor armazenado na variável **nome** é igual a Jose. Se o usuário digitar qualquer outro nome, ou inclusive jose ou José o resultado da comparação é falso.

Então o **.equals()** é utilizado como: <nomedavariável>.equals (valor a comparar).

VOCÊ NO COMANDO

Como poderíamos elaborar um programa que faça a leitura de um nome de usuário e uma senha e libere o acesso ao aplicativo somente se ambos estiverem corretos?

Após elaborar o seu programa, confira no quadro a seguir se você codificou a situação corretamente.

```
import javax.swing.JOptionPane;

public class LoginSenha {

    public static void main(String[] args) {
        // Login e senha
        // login = aluno
        // senha = aluno

        //declaração de variáveis
        String login, senha; // variáveis para armazenar o login e senha

        //entrada de dados
        login = JOptionPane.showInputDialog("Entre com o Login");
        senha = JOptionPane.showInputDialog("Entre com a senha");

        if( login.equals("aluno") && senha.equals("aluno")) {
            JOptionPane.showMessageDialog(null, "Acesso liberado");
        }else {
            JOptionPane.showMessageDialog(null, "Login ou senha
incorretos");
        } // fim do if

    } //fim do main
} // fim da classe
```



VOCÊ NO COMANDO

Elabore um fluxograma e um programa em Java que leia um número e compare se ele é maior ou igual a 100.

Após a elaboração do seu fluxograma, pseudocódigo e programação em Java, confira com as resoluções a seguir:

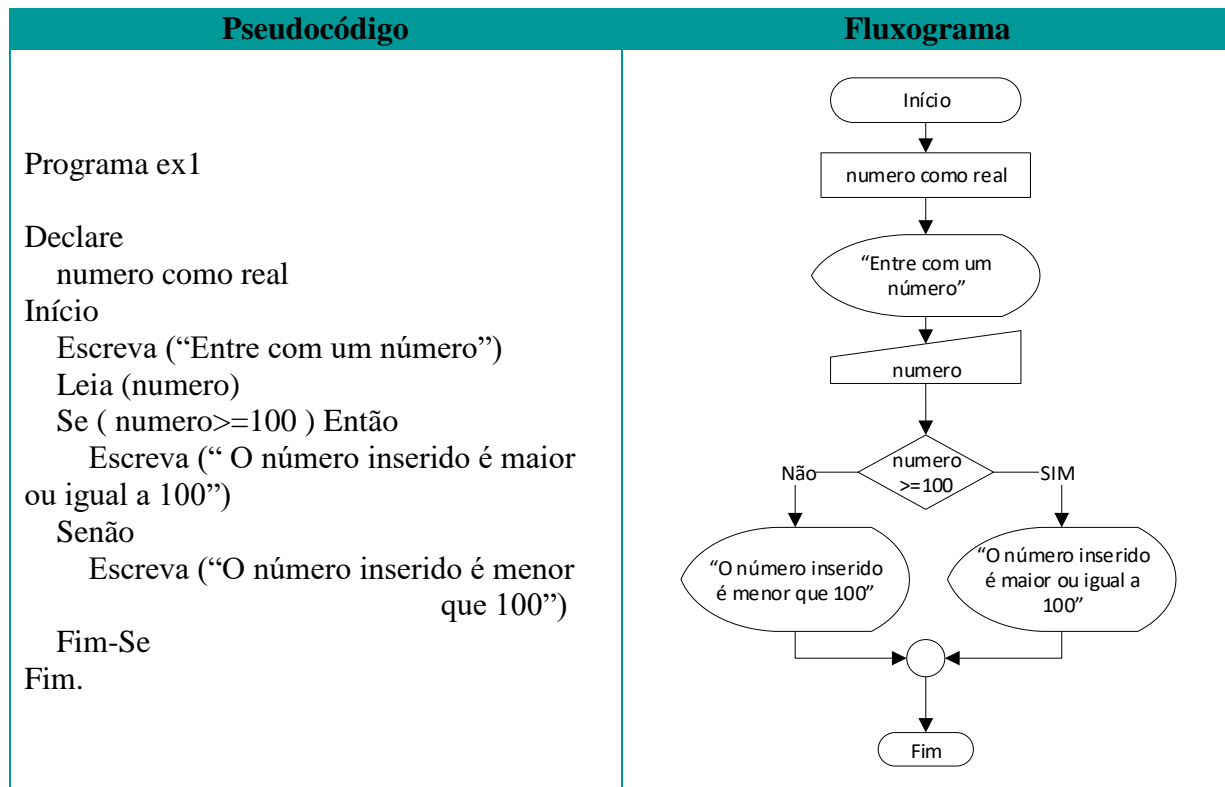


Imagem 14: GEEaD – Representação de Pseudocódigo e Codificação Java – Comando “Se..Senão” para verificar se o número digitado é menor do que 100 ou se o número digitado é maior ou igual a 100.

Codificação em Java:

```

public class if_Ex1 {

    public static void main(String[] args) {
        // exercício 1

        //declaração de variáveis
        double numero;
        String aux;

        //entrada de dados
        aux = JOptionPane.showInputDialog("Entre com um número");
        numero = Double.parseDouble(aux);

        //processamento e saída
        if (numero >= 100) {
            JOptionPane.showMessageDialog(null, "O número inserido é maior ou igual que 100");
        } else {
            JOptionPane.showMessageDialog(null, "O número inserido é menor que 100");
        }

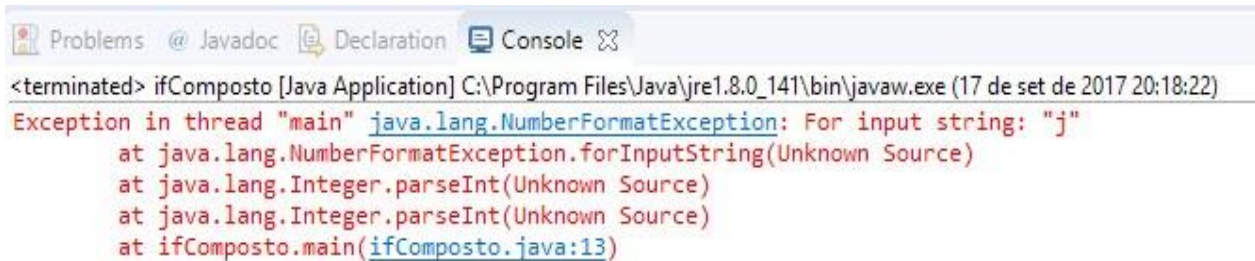
    }

}

```

Tratamento de erros com o comando Try-Catch

Até agora não foi realizado nenhum tratamento de erros quando o usuário insere algo errado no sistema como, por exemplo, inserir um caractere no lugar de um número. Quando isso acontece sem o tratamento de erro, do ponto de vista do usuário, o programa simplesmente fecha. Do ponto de vista do programador, uma mensagem de erro é gerada no IDE (ambiente de desenvolvimento integrado) para alertá-lo. A figura a seguir ilustra um erro de entrada de dados:



```
<terminated> ifComposto [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (17 de set de 2017 20:18:22)
Exception in thread "main" java.lang.NumberFormatException: For input string: "j"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at ifComposto.main(ifComposto.java:13)
```

Imagem 20: Erro de entrada de dados: Exception in thread "main"
 java.lang.NumberFormatException: For unout string: "j"
 at java.lang.NumberFormatException.forInputString(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at ifComposto.main(ifComposto.java:13)



Parece uma coisa desnecessária realizarmos tratamento de erros à primeira vista. Mas reflita: quantas vezes não ficamos com raiva quando, ao utilizarmos algum software ou app ele simplesmente fecha sem dar nenhuma explicação? Temos que pensar sempre no usuário do programa.

Mas, o problema realmente é do usuário que fica sem saber o que aconteceu. Com certeza, algum programa de computador ou app de celular já fechou inesperadamente sem nenhum aviso de erro e perdemos o que já estava sendo feito. Isso dá uma raiva imensa.

Para evitar ou minimizar aborrecimentos, no Java existem várias técnicas e rotinas de tratamento de erros. Vamos apresentar aqui o comando try-catch. Ele consiste em capturar erros na conversão de tipos na entrada de dados para os nossos programas. O comando trata mais tipos de erro, mas não é o escopo desta explicação. A sintaxe é:

```
try{
    comando(s);
} catch (NumberFormatException e){
    Comando(s);
}
```

Vamos explicar com um exemplo:

```
import javax.swing.JOptionPane;

public class tryCatch {

    public static void main(String[] args) {
        // declaração de variáveis
        int numero=0;
        String aux;

        //entrada de dados
        try {
            aux = JOptionPane.showInputDialog("Entre com um número inteiro");
            numero = Integer.parseInt(aux);
            JOptionPane.showMessageDialog(null, "O número inserido foi " +
numero);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "Entre somente com um número
Inteiro",
                "E R R O", JOptionPane.ERROR_MESSAGE);
        } //fim do try-catch
    } //fim do método main
} //fim da classe
```

O try-catch funciona da seguinte forma:

Na linha 11, o comando **try** tenta executar os comandos das linhas 12 a 14 que estão dentro das chaves, que é fechada na linha 15. Caso não consiga, ele executa a cláusula **catch (NumberFormatException e)** e os comandos dentro das chaves das linhas 15 a 18. Isto exibirá uma mensagem de erro.

Caso o usuário realmente digite um número como o solicitado pelo programa, a seguinte mensagem será exibida: “O número inserido foi <numero>” como nas figuras a seguir:

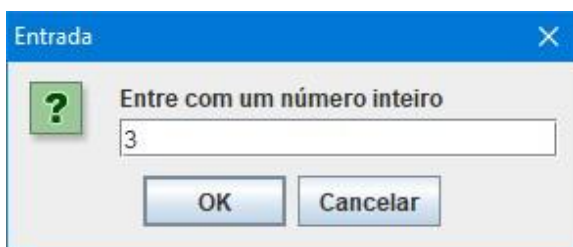


Imagem 06: Caixa de entrada de dados com a mensagem “Entre com um número inteiro”, logo abaixo uma caixa de entrada de dados com o número 3. À esquerda tem o botão “OK” e à direita o botão “Cancelar”.

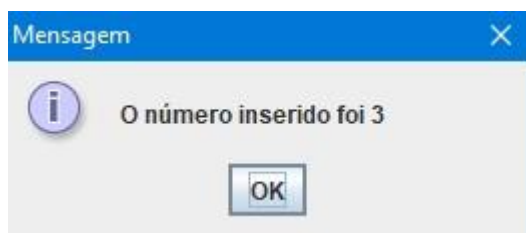


Imagem 07: Caixa de mensagem informativa com a mensagem “O número inserido foi 3” e um botão “OK” centralizado na caixa.

Quando um erro de inserção de dados ocorre e, por exemplo, um caractere ou número real é inserido a seguinte mensagem de erro aparece:

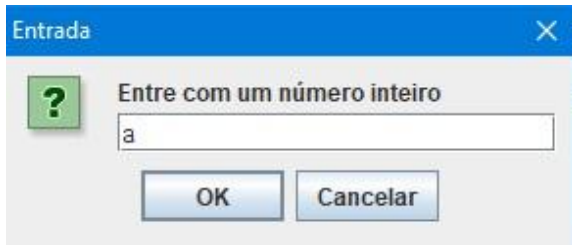


Imagem 08: Caixa de entrada de dados com a mensagem “Entre com um número inteiro”, logo abaixo uma caixa de entrada de dados com o a letra a em minúsculo. À esquerda tem o botão “OK” e à direita o botão “Cancelar”.

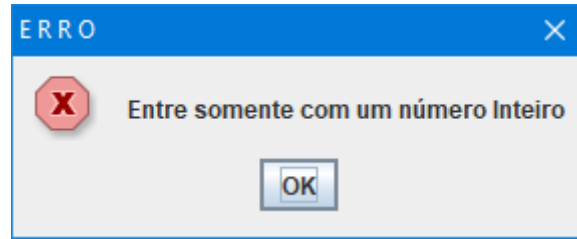


Imagem 09: Caixa de mensagem de erro com a mensagem “O número inserido foi 3 e um botão “OK” centralizado na caixa.

Note que o try-catch capturou o erro de conversão, afinal o caractere “a” não é um número inteiro e exibiu a mensagem de erro correspondente.

Um detalhe interessante: não sei se repararam, mas a caixa de diálogo da **Erro! Fonte de referência não encontrada**. é bem diferente da **Erro! Fonte de referência não encontrada**. Isso porque durante a escrita da caixa de diálogo de saída, esta foi formatada. Vamos analisar o comando da linha 16 e 17:

```
JOptionPane.showMessageDialog(null, "Entre somente com um número Inteiro",
    "E R R O", JOptionPane.ERROR_MESSAGE);
```

O comando **JOptionPane.showMessageDialog(null, “Entre com um número Inteiro”** exibe a caixa de diálogo que já conhecemos. Note que o número de argumentos do comando aumentou:

```
,"E R R O", JOptionPane.ERROR_MESSAGE);
```

É isso que inclui a formação da caixa de diálogo. A parte em **vermelho** (“E R R O”), insere o título da janela e a parte em **verde** (JOptionPane.ERROR_MESSAGE) altera o ícone para uma cruz vermelha para indicar erro.



Uma observação: os argumentos são sempre separados por vírgula

Exemplo:

Se quisermos exibir uma mensagem de alerta para o usuário com um ponto de exclamação o código será:

```
import javax.swing.JOptionPane;

public class mensagem_de_alerta {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Mensagem de Alerta",
            "Alerta", JOptionPane.WARNING_MESSAGE);
    }
}
```


Perceba que o título da janela foi definido como Alerta.

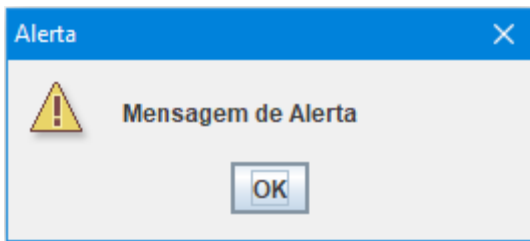


Imagem 10: Caixa de mensagem de alerta e um botão “OK” centralizado na caixa.

VOCÊ NO COMANDO

Acabamos de ver como podemos modificar a exibição de uma caixa de diálogo de saída de dados. Explore um pouco na IDE Eclipse algumas outras personalizações desta caixa de diálogo. Utilize a função de auto completar comandos da IDE. Para usar esta função basta digitar o início de um comando e pressionar as teclas Control (CTRL) + barra de espaços.