

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
LÓGICA DE PROGRAMAÇÃO

Autores:

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

Revisão Técnica:

Kelly Dal Pozzo de Oliveira

Revisão Gramatical:

Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação:

Flávio Biazim

AGENDA 5

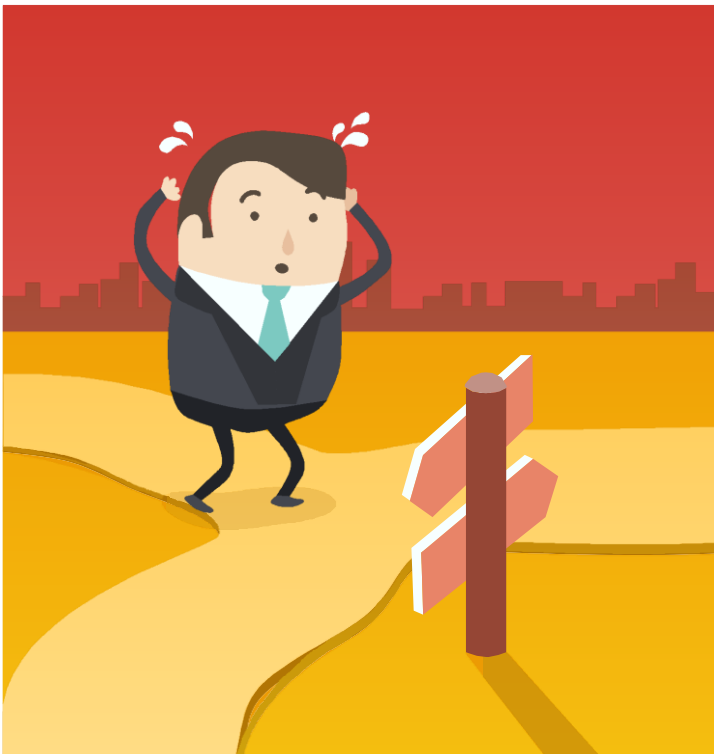
ESTRUTURAS DE DECISÃO





MERGULHANDO NO TEMA...

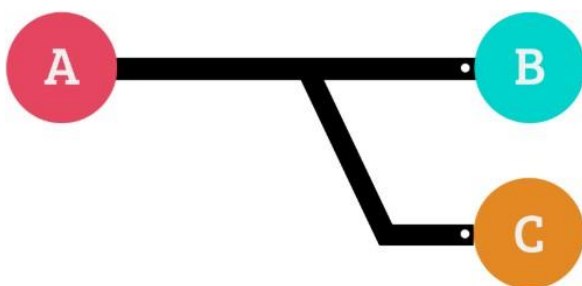
Estruturas de Decisão



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	<pre> graph TD Start(()) --> Cond{condição} Cond -- SIM --> Com[Comando(s)] Cond -- NÃO --> End(()) Com --> End End --> Exit(()) </pre>	<pre> if (condition){ {comando(s); } </pre>

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	<pre> graph TD Start(()) --> Dec{idade >=18} Dec -- SIM --> Out([Maior de idade]) Dec -- NÃO --> Conn(()) Out --> Join(()) Conn --> Join Join --> End(()) </pre>	if (idade >=18){ System.out.println ("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 **System.out.println** e finalmente o **Fim-Se** é substituído pelo **}**.

Realizando a codificação em Java de um programa completo teremos:

```

import java.util.Scanner;
public class IfSimples {

    public static void main(String[] args) {
        //declaração de variáveis
        int idade; // armazena a idade
        Scanner leitor = new Scanner(System.in);

        //entrada de dados
        System.out.println("Entre com a sua idade");

        idade = leitor.nextInt();

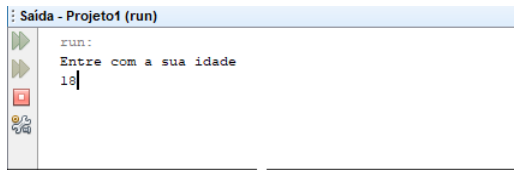
        //Decisão
        if (idade >=18) {
            System.out.println("Maior de Idade");
        } //fim do if
    } //fim do main

} //fim da classe

```

Explicando a Estrutura de decisão, o comando de decisão if (idade \geq 18) irá executar o que estiver dentro das chaves, 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.

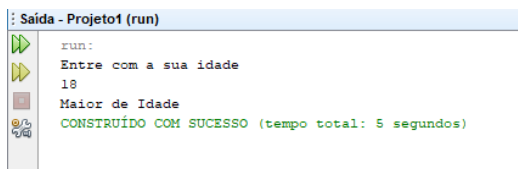
Se, ao executarmos o programa, digitarmos a idade de 18 anos, por exemplo:



```
run:
Entre com a sua idade
18
```

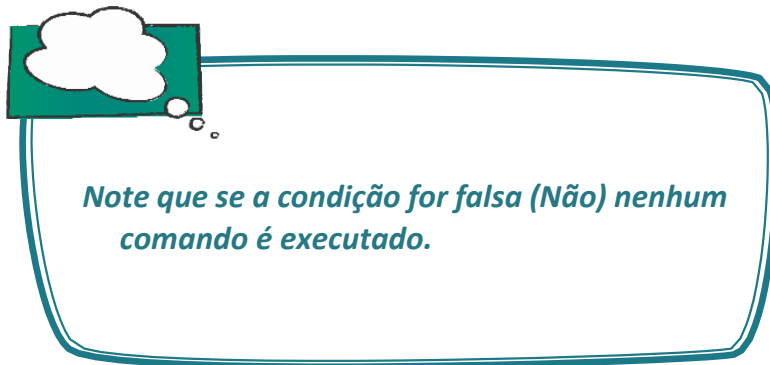
Imagem 02: Console para Entrada de Dados com a mensagem “Entre com a sua idade”. No console foi digitado o número 18.

Teremos o seguinte retorno:



```
run:
Entre com a sua idade
18
Maior de Idade
CONSTRUÍDO COM SUCESSO (tempo total: 5 segundos)
```

Imagem 03: Resultado exibido no Console após informando que a idade digitada é “Maior Idade”.



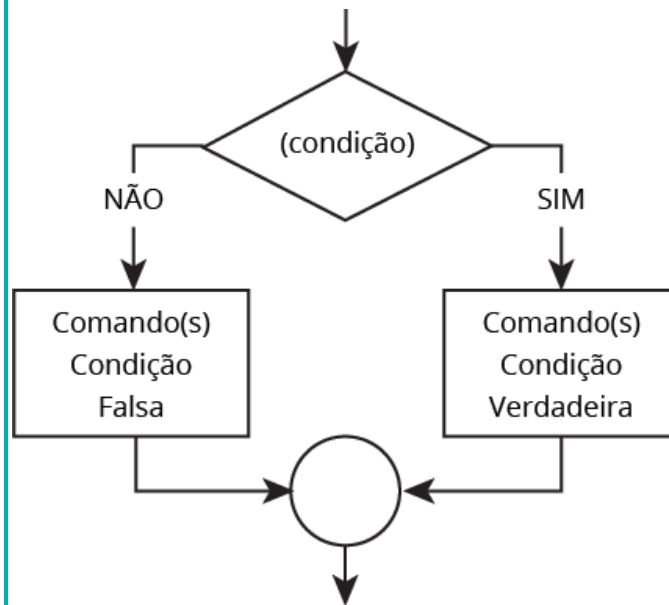
Estrutura de decisão IF-else (se...Senão...Fim se)

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

SE (condição) **Então**
 {comando(s) condição verdadeira}
Senão
 {Comando(s) condição falsa}
Fim-Se

FLUXOGRAMA



JAVA

```

if (condition){
    {comando(s)
    condição verdadeira};
}
else {
    {comando(s)
    condição falsa};
}
  
```

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	<pre> graph TD A{idade >=18} -- NÃO --> B([Menor Idade]) A -- SIM --> C([Maior Idade]) B --> D(()) C --> D style D fill:none,stroke:none </pre>	<pre> if (idade >=18){ System.out.println("Maior de idade"); } else { System.out.println ("Menor de idade"); } </pre>

Imagem 09: GEEaD – Representação de Pseudocódigo, Fluxograma e Codificação Java – Comando “Se..Senão” 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 java.util.Scanner;

public class IfComposto {

    public static void main(String[] args) {
        //declaração de variáveis
        int idade; // armazena a idade
        Scanner leitor = new Scanner(System.in);

        //entrada de dados
        System.out.println("Entre com a sua idade");

        idade = leitor.nextInt();

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

} //fim da classe
          
```

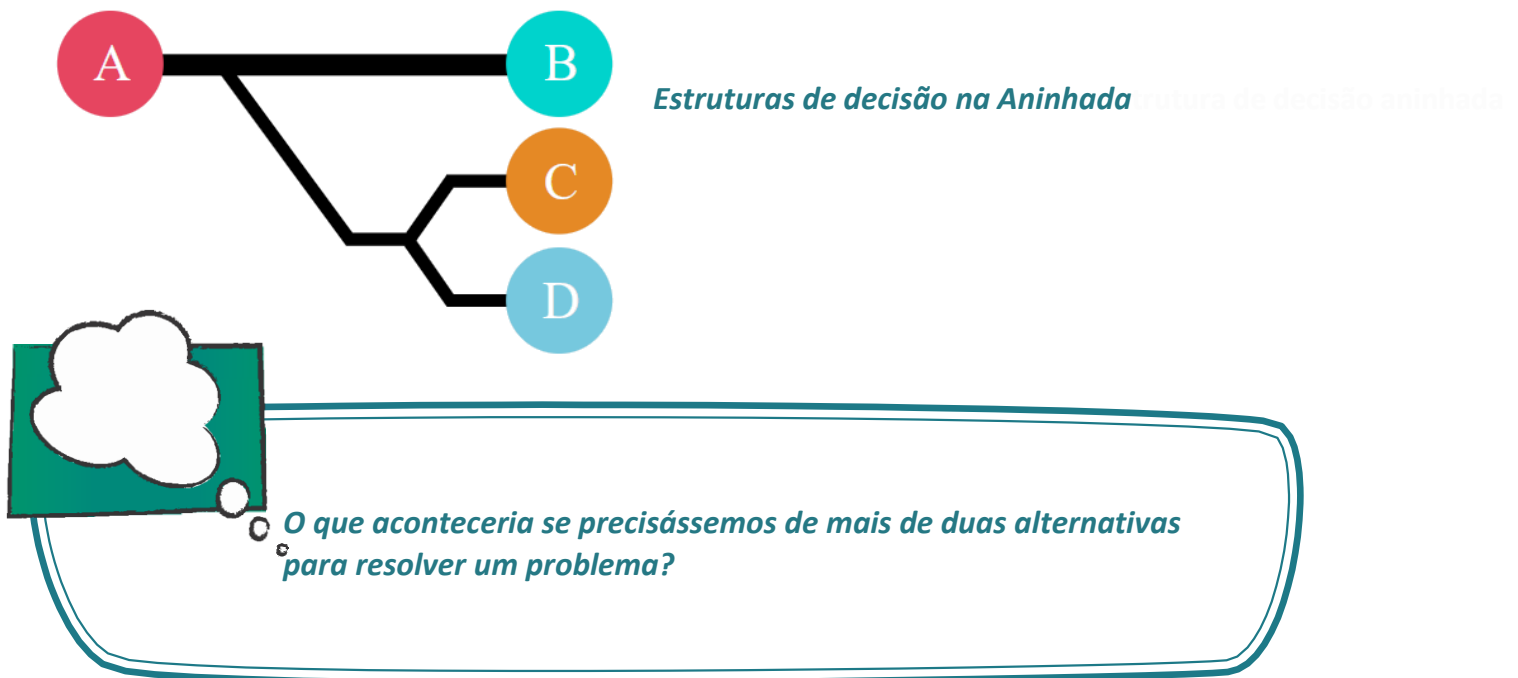
Se digitarmos 17 no Console, o resultado será:

```

Saída
Projeto1 (run) x Projeto1 (run) #2 x
run:
Entre com a sua idade
17
Menor de Idade
CONSTRUÍDO COM SUCESSO (tempo total: 6 segundos)

```

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:

PSEUDOCÓDIGO	FLUXOGRAMA	JAVA
SE (idade >=18) Então SE (idade = 18) Então Escreva ("Igual a 18") Senão Escreva ("Maior de 18") Fim-Se Senão Escreva ("Menor de idade") Fim-Se	<pre> graph TD A{idade >= 18} -- NÃO --> B([Menor Idade]) A -- SIM --> C{idade = 18} C -- NÃO --> D([Maior de 18]) C -- SIM --> E([Igual a 18]) B --> F(()) D --> F E --> F F --> G[] </pre>	<pre> if (idade >=18){ if (idade==18){ System.out.println("igual a 18"); } else { System.out.println("Maior de 18"); } } else { System.out.println("Menor de idade"); } </pre>

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 afirmativo 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
        Scanner leitor = new Scanner(System.in);
        //entrada de dados
        System.out.println("Entre com a sua idade");
        idade = leitor.nextInt();

        //Decisão
        if (idade >=18) { // primeiro if
            //comandos se a condição for verdadeira
            if (idade == 18) { // segundo if
                System.out.println("igual a 18");
            }else {
                System.out.println("Maior de 18");
            }//fim do segundo if
        } else {
            //comandos se a condição for falsa
            System.out.println("Menor de Idade");
        } // fim do primeiro if
    } //fim do main

} //fim da classe

```

Aprimorando a comunicação com o usuário

Antes de mais nada, os programas devem ser fáceis de se utilizar e os usuários devem fazer isso intuitivamente. Para tanto, todo programador deve atentar-se à interface com o usuário. Vamos aprender uma maneira alternativa de entrada de saída de dados mais elegante para utilizarmos em nossos futuros programas em Java. Até aqui utilizamos o que chamamos de modo console, ou seja, uma tela de terminal que não aceitava elementos gráficos para interagirmos com o programa.

```
Entre com um nome:
José
O nome é: José
```

Imagem 31: Terminal de saída de dados – janela onde será exibida a execução da aplicação.

VOCÊ NO COMANDO

Um programador deve sempre tentar desenvolver uma interface com o usuário que seja simples de ser utilizada. Contudo, essa interface deve agradar ao cliente que está contratando os seus serviços. Pensando como programador, antes de prosseguir a leitura, reflita como isso pode interferir na prática de programação.

Atualmente, vivemos em um mundo onde os elementos gráficos predominam em todos os aspectos de comunicação. Portanto, será apresentado um modo gráfico utilizando as janelas do sistema operacional para realizarmos a entrada e a saída de dados.

Entrada de dados



Para realizarmos a entrada de dados para um programa, utilizaremos o comando **JOptionPane** do Java.

A sintaxe do comando é a seguinte:



Imagem 32: Sintaxe do comando `JOptionPane` para entrada de dados. Composição da sintaxe `JOptionPane.showInputDialog(mensagem);`. `JOptionPane` – é a biblioteca responsável pela interação com o Sistema Operacional. `showInputDialog` – indica que o comando a ser utilizado será uma entrada de dados do sistema. `(mensagem)` – é a mensagem a ser exibida para o usuário.

Exemplo de aplicação em um programa:

```

1- import javax.swing.JOptionPane;
2-
3- public class JoptionPane {
4-
5- public static void main(String args[]) {
6-
7- String entrada; //variável de entrada
8- entrada = JOptionPane.showInputDialog("Entre com um nome");
9-
10- }
11-
12- }

```

Imagem 33: Mostra um exemplo de aplicação de um programa em Java.

Na linha 7, realizamos a declaração da variável entrada do tipo String que armazenará o conteúdo inserido pelo usuário.

Na linha, 8 a variável entrada recebe o conteúdo do comando `JOptionPane.showInputDialog("Entre com um nome")`;

Vamos analisar como esta linha se comporta. Este comando solicita ao usuário que digite algum dado para o sistema. Mas qual dado é este? No nosso exemplo, o dado solicitado é o nome – Entre com um nome.

O resultado para o usuário será:

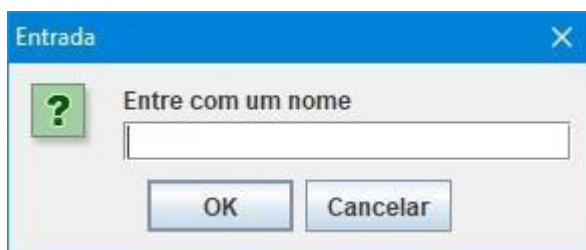


Imagem 34: Tela de entrada de dados – com uso do modo gráfico. Essa imagem mostra uma caixa de entrada de dados com uma mensagem: “Entre com um nome”, e dois botões: “Ok” à esquerda e “Cancelar” à direita.

Saída de dados



Um resultado muito mais bonito esteticamente que o modo console.

Para realizarmos a saída dos dados de um programa também utilizaremos o comando **JOptionPane**, mas com a seguinte sintaxe:



Imagem 35: Sintaxe do comando `JOptionPane` para saída de dados. Composição da sintaxe `JOptionPane.showMessageDialog(null, mensagem);`
`JOptionPane` – é a biblioteca responsável pela interação com o Sistema Operacional. `showMessageDialog` – indica que o comando a ser utilizado será uma saída de dados do sistema. `(null, mensagem)` – é a mensagem a ser exibida para o usuário.

Exemplo de aplicação em um programa:

```

1- import javax.swing.JOptionPane;
2-
3- public class JOptionPaneShow {
4-
5- public static void main(String args[]) {
6-     JOptionPane.showMessageDialog(null, "Saída de dados");
7-
8- }
9-
10- }
```

Imagem 36: Tela de exemplo de codificação do comando `JOptionPane` para saída de dados

Na linha 6, o comando `JOptionPane.showMessageDialog (null, "saída de dados");` faz a saída da mensagem para o usuário. O primeiro argumento sempre será `null`, seguido pela mensagem que queremos exibir para o usuário – Saída de dados. O resultado será:

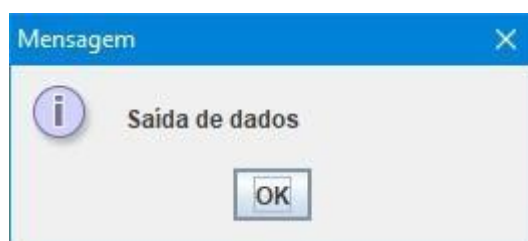


Imagem 37: Caixa de mensagem de informação, escrito "Saída de dados" e um botão de Ok centralizado.

Agora, sabendo como realizar a entrada e a saída de dados de forma gráfica, conseguimos fazer um programa que leia e exiba dados para o usuário em modo gráfico, como no exemplo a seguir:

```

1- import javax.swing.JOptionPane;
2-
3- public class JoptionPane {
4-
5- public static void main(String args[]) {
6-
7- String nome; //variável nome do tipo String
8-
9- //entrada de dados
10-     nome = JOptionPane.showInputDialog("Entre com o seu
        nome");
11-
12-     //saída de dados
13-     JOptionPane.showMessageDialog(null,"O seu nome é " +
        nome);
14- }
15-
16- }

```

Imagem 38: Codificação para leitura e exibição de dados em modo gráfico

Neste exemplo apresentado, o programa exibe uma janela pedindo o nome do usuário na linha 10 e posteriormente exibe o nome digitado na linha 13. Resultado:

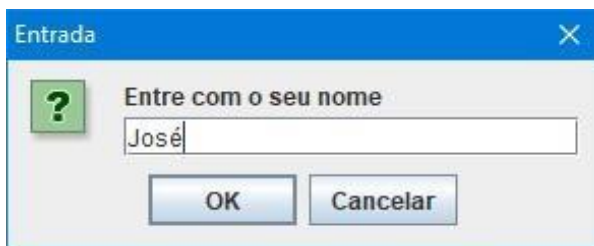


Imagem 39: Caixa de Entrada de Dados do tipo "Question" com a mensagem "Entre com o seu nome". Na caixa de entrada de dados foi digitado o texto "José". À esquerda tem o botão "OK" e à direita o botão "Cancelar".

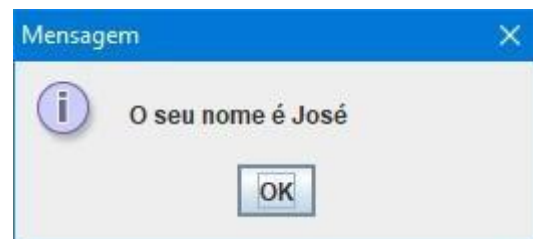


Imagem 40: Caixa de Saída de Dados informando que o nome digitado foi "José" e um botão de "Ok".

Conversão de tipos

O comando `JOptionPane.showInputDialog` sempre gera uma saída de dados do tipo `String` (sequência de caracteres alfanuméricos). Portanto, se utilizarmos tipos de variáveis diferentes como, por exemplo, inteiro, devemos fazer a conversão de tipos antes de armazenarmos na sua variável correspondente conforme o exemplo a seguir:


```

1- import javax.swing.JOptionPane;
2-
3- public class JOptionPaneCast {
4-
5-     public static void main(String args[]) {
6-         String auxiliar; //variável auxiliar
7-         int numeroInteiro ;
8-         double numerodouble;
9-         float numerofloat;
10-
11-         // entrada de dados salvando na variável auxiliar (string)
12-         auxiliar = JOptionPane.showInputDialog("Entre com um número inteiro");
13-
14-         //conversão do tipo string para inteiro - Integer.parseInt
15-         numeroInteiro = Integer.parseInt(auxiliar);
16-         numerodouble = Double.parseDouble(auxiliar);
17-         numerofloat = Float.parseFloat(auxiliar);
18-
19-         //mensagem de saída
20-         JOptionPane.showMessageDialog(null,"O número inteiro é " + numeroInteiro);
21-         JOptionPane.showMessageDialog(null,"O número double é " + numerodouble);
22-         JOptionPane.showMessageDialog(null,"O número float é " + numerofloat);
23-
24-     }
25-
26- }

```

Imagem 41: Codificação de conversão de dados – exemplo de uso do comando JOptionPane.showInputDialog.



Um tipo nada mais é do que o conteúdo que uma variável consegue armazenar. Um tipo String pode armazenar caracteres e números. Um tipo "int", números inteiros e os tipos "double" e "float", números reais.

No exemplo da imagem acima, a entrada de dados feita na linha 12 é salva na variável auxiliar que é do tipo String. Porém, nesse exemplo, estamos trabalhando com números e um dado do tipo String para armazenar textos. Para isto, devemos fazer a conversão de tipo (cast em inglês), ou seja, temos que realizar a conversão de um tipo de dado para outro. Ex.: de String para int, de String para double etc. O Java possui comandos específicos para executar o cast.

A sintaxe para a conversão de um tipo **String para Inteiro** é:

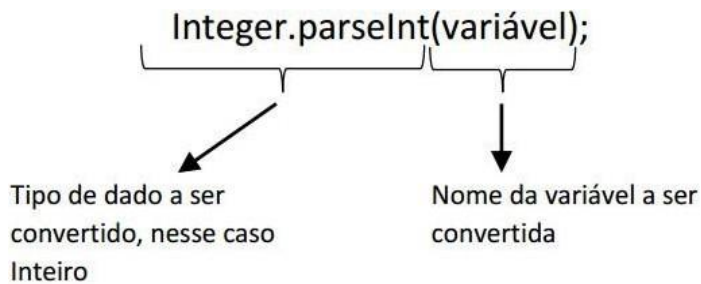


Imagem 42 sintaxe para a conversão de um tipo String para inteiro. Composição da sintaxe `Integer.parseInt(variável);`
`Integer.parseInt` – é o tipo de dado a ser convertido, nesse caso, inteiro.
`(variável);` - é o nome da variável a ser convertida.

A sintaxe para a conversão de um tipo **String para Double** é:

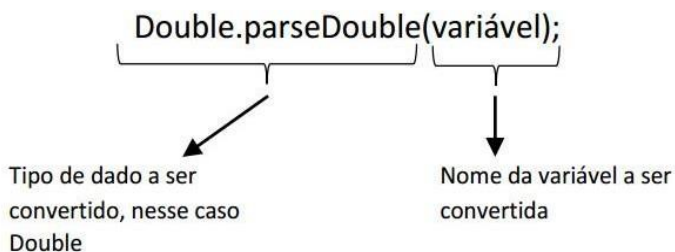


Imagem 43: sintaxe para a conversão de um tipo String para Double.
 Composição da sintaxe `Double.parseDouble(variável);`
`Double.parseDouble` – é o tipo de dado a ser convertido, nesse caso, Double.
`(variável);` - é o nome da variável a ser convertida.

A sintaxe para a conversão de um tipo **String para Float** é:

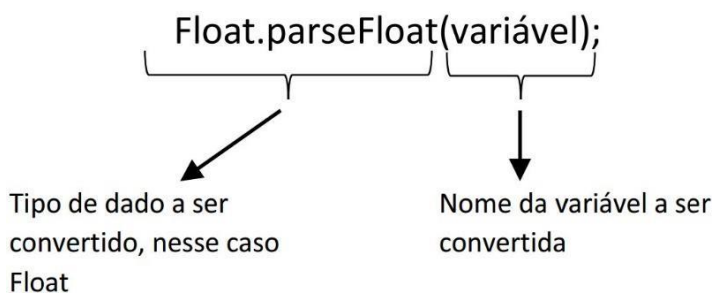


Imagem 44: sintaxe para a conversão de um tipo String para Float.
 Composição da sintaxe `Float.parseFloat(variável);`
`Float.parseFloat` – é o tipo de dado a ser convertido, nesse caso, Float.
`(variável);` - é o nome da variável a ser convertida.

Conhecendo a sintaxe do comando; na linha 15, a variável **numeroInteiro** recebe o resultado da conversão do valor da variável **auxiliar** de String para Inteiro, assim como as linhas 16 e 17 realizam o cast para double e float, respectivamente.

Um aspecto interessante na linguagem Java é que no comando `JOptionPane.showMessageDialog`, não precisamos realizar a conversão de tipos para o `String`. O Java, automaticamente, converte tipos numéricos para `String` antes de exibir a mensagem para o usuário.

Uma observação: existem outras formas de realizarmos a conversão de tipos e para mais tipos de dados, porém não serão apresentadas aqui nesse momento.

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. 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. 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.

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

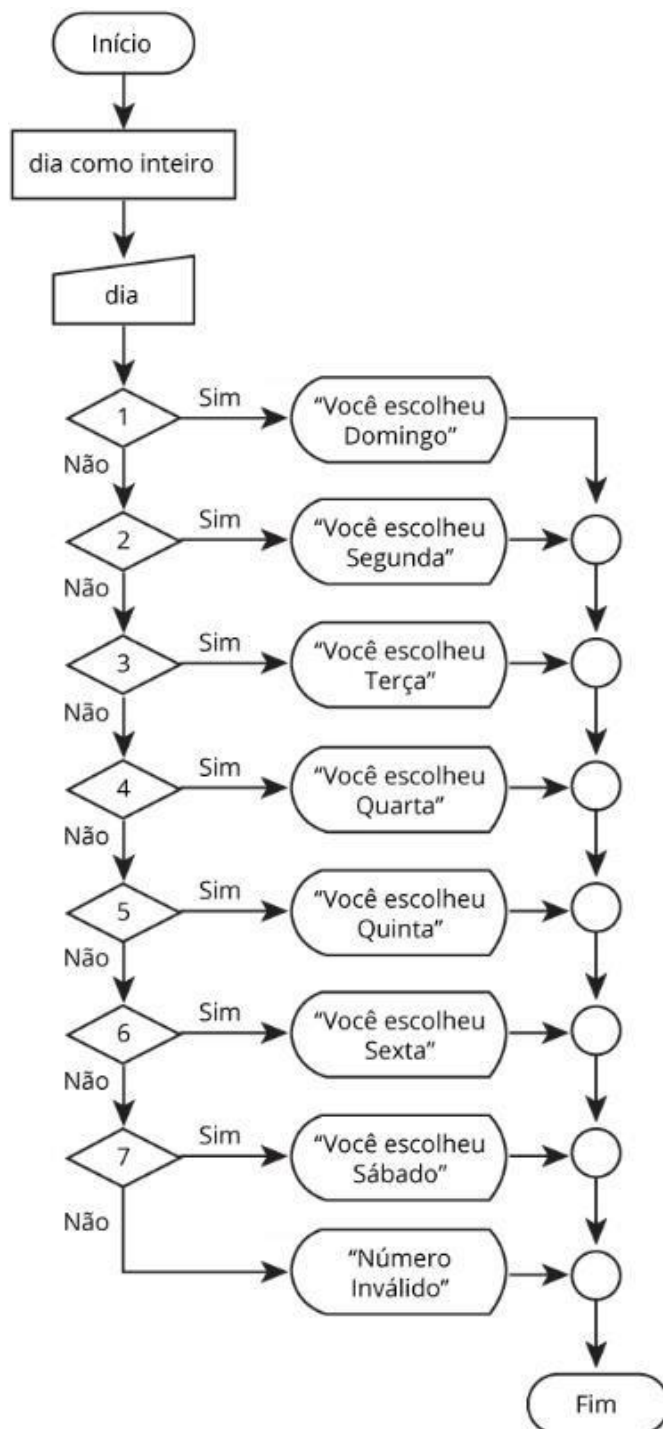
Fim-se

Fim-Se

Fim-Se

Fim-Se

Fim.



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
```

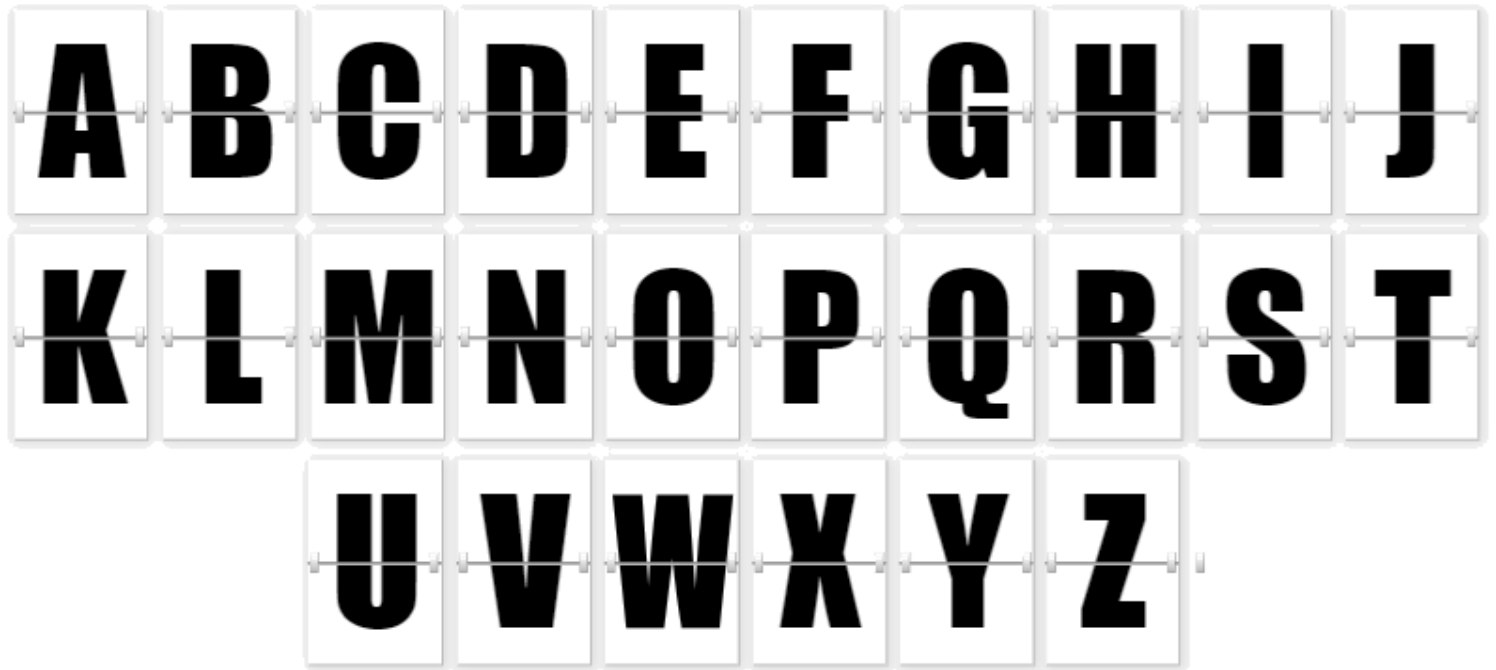


VOCÊ NO COMANDO

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 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:

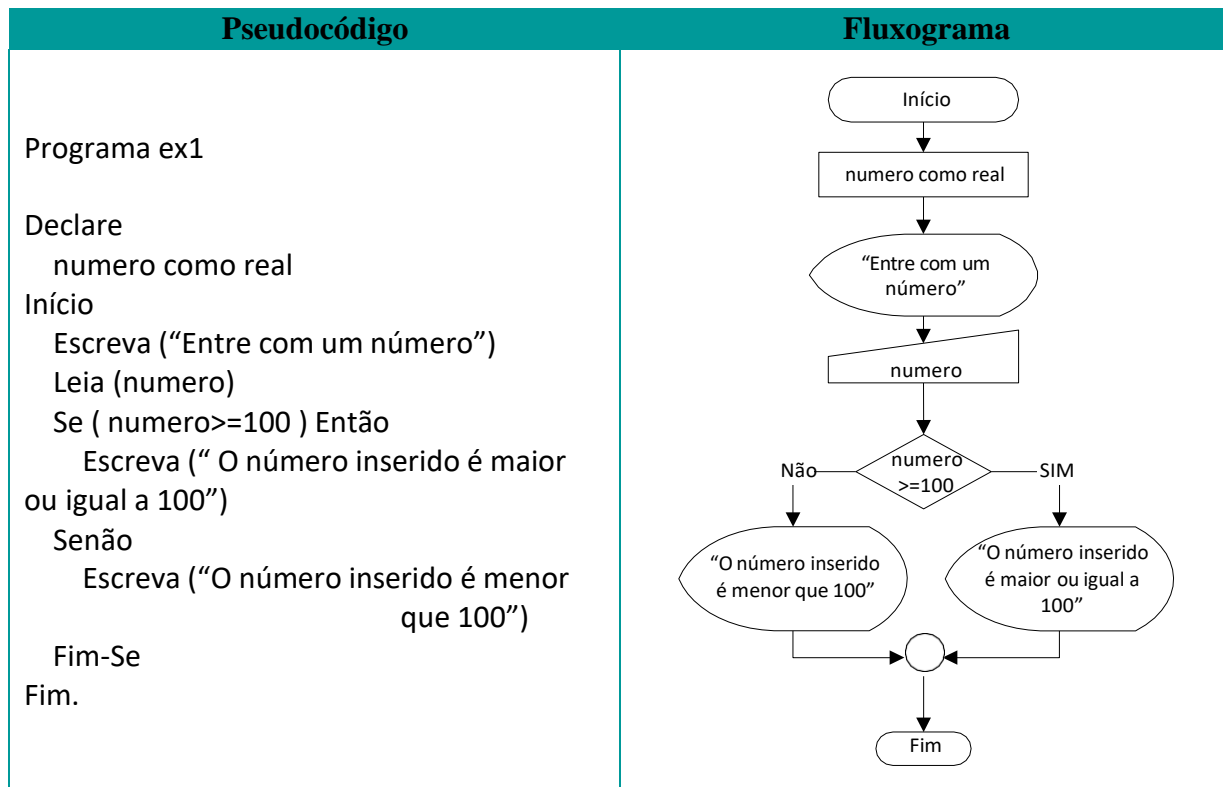


Imagem 14: GEEaD – Representação de Pseudocódigo e Codificação Java – Comando “Se..Senão” para verificar se o número digitado é menor do que 100 ou se o número digitado é maior ou igual a 100.

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");
        }

    }

}

```