

Agenda 04: Estruturas de Decisão

Conceitos trabalhados:

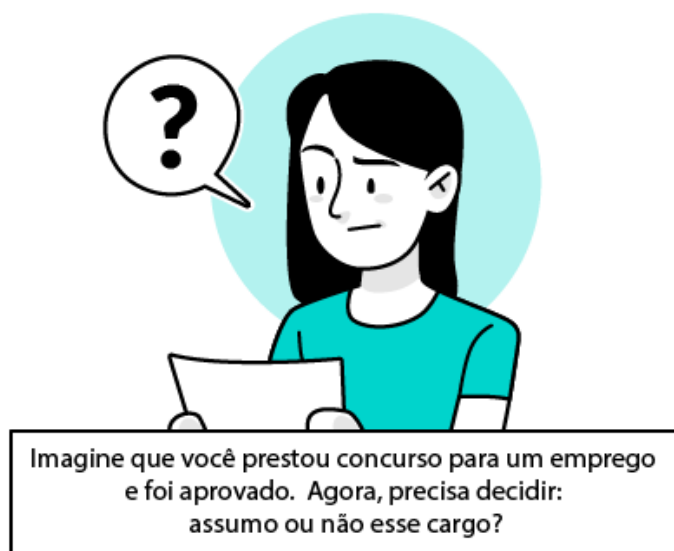
Sumário

- Momento de Reflexão
- Por que Aprender?
- Para Começar o Assunto
- Mergulhando no Tema
- Ampliando Horizontes
- Resumindo o Estudo

Momento de Reflexão

Olá, estudante ;)

Boas-vindas à quarta agenda do Módulo 1, que marca a metade do seu percurso neste módulo! Nesta agenda, você estudará as **Estruturas de Decisão**, conceito fundamental para o desenvolvimento de qualquer software. Há momentos na vida em que precisamos tomar algumas decisões, não é? E sabemos que dependendo da decisão que tomamos seguimos para um caminho ou outro na vida:





Você percebeu que a partir de uma tomada de decisão **optamos por fazer ou não fazer** determinadas ações, escolhemos viver ou não determinadas situações. Bem, assim também acontece quando fazemos um programa de computador ou aplicativos para telefones celulares inteligentes, os chamados Apps.

Em alguns momentos, nas rotinas dos programas que desenvolvemos, o computador deverá tomar decisões **de acordo com as condições estabelecidas**. E esse será o tema de estudo desta agenda, em que você aprenderá a utilizar as Estruturas de Decisão em um programa de computador, permitindo que o programa execute diferentes tipos de procedimentos baseados em uma determinada decisão.

Vamos lá?

Por que Aprender?

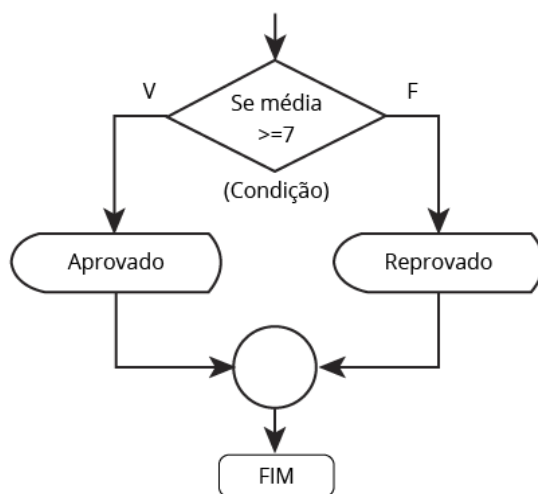
Você já sabe que um programa é uma **lista ordenada de comandos que são executados sequencialmente**. A menos que seja solicitado o contrário, um programa executa as instruções do início até o fim. Mas, conforme aumenta a complexidade do programa e da tarefa que precisa executar, ele precisará tomar decisões para alterar a ordem desse processamento sequencial. É por este motivo que você estudará, nesta agenda, as noções de estruturas de dados **Se., Senão.. e Fim-se.**

Para Começar o Assunto



Na agenda anterior, você viu o diálogo de um aluno pedindo a um computador que calculasse sua média escolar. Fazer um programa que faça esse cálculo e apresente o resultado em tela é tranquilo, não é mesmo?

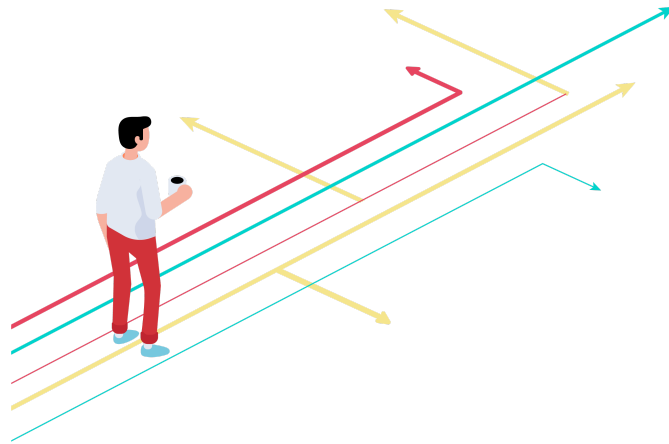
Mas imagine que este programa também deva escrever na tela se o aluno está aprovado ou reprovado de acordo com a média calculada. Neste caso, em determinado momento, será necessário que o programa **verifique a média calculada e tome a decisão de escrever na tela: Aprovado ou Reprovado.**



Se a média for maior ou igual a 7 (ou seja, se a condição for verdadeira), o aluno foi aprovado, senão (se a condição for falsa), o aluno foi reprovado. Entendeu? Os materiais a seguir detalharão melhor essa estrutura!

Mergulhando no Tema

Estruturas de decisão



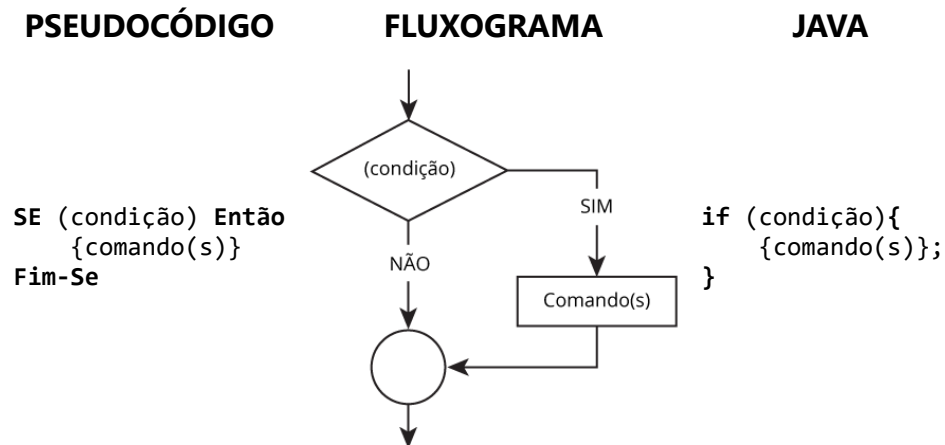
Até agora, trabalhamos somente com programas que efetuam tarefas simples, como a realização de entrada e saída de dados e pequenos cálculos matemáticos. Você deve ter percebido que os algoritmos criados até aqui não possuem **poder de decisão**.

Em outras palavras, eles sempre executam as **mesmas tarefas**, independentemente dos resultados obtidos. Mas, na vida real precisamos tomar decisões que muitas vezes são difíceis e que podem alterar o rumo de nossas vidas. Quando falamos de programas, ocorre a mesma coisa.



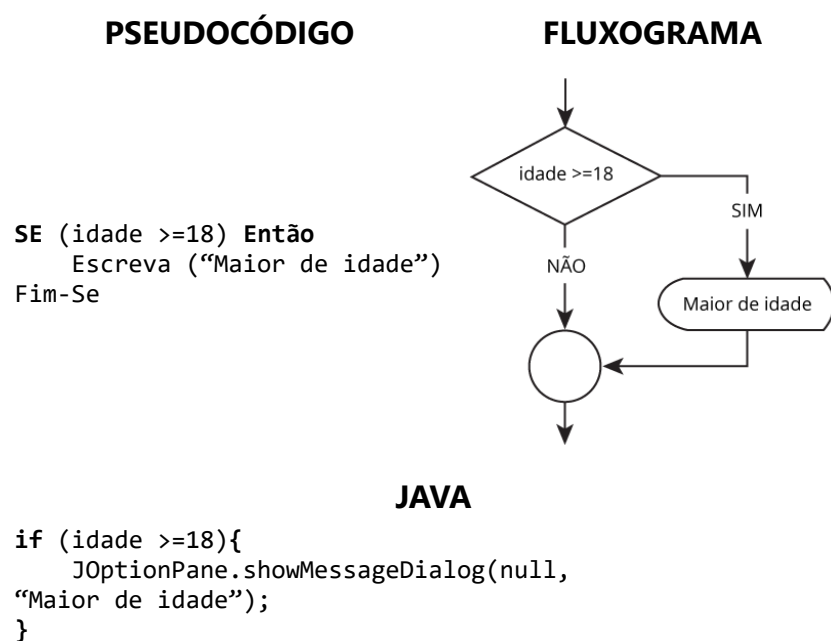
Em programação, chamamos essas decisões de **Estrutura de Decisão**. No Java, chamamos de comando **If**. Estruturas de Decisão **if** (Se... Fim.)

As Estruturas de Decisão ou Testes Condicionais nos permitem executar um conjunto diferente de comandos dependendo do resultado de um teste utilizando **operadores relacionais**. Este resultado pode ser verdadeiro ou falso conforme indicado na tabela a seguir:



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:



O comando condicional **SE** testa a condição **idade >=18**. Ou seja, se a idade for maior ou igual a 18 anos **ENTÃO** a condição é verdadeira (segue-se o caminho “sim”). Desse modo, será executado o comando **Escreva** (“**Maior de idade**”). Caso contrário (se a idade for menor que 18 anos), nada será feito (**Fim-Se**). Observando o fluxograma, você compreenderá com mais facilidade o que ocorre.

Você deve ter reparado, com base na sintaxe apresentada na tabela acima, que na programação Java a palavra **SE** é substituída pelo **if** e que a palavra **Então** é substituída por **{**. O comando **Escreva** é substituído por **JOptionPane.showMessageDialog** e, finalmente, o **Fim-Se** é substituído por **}**.

Agora, veja a codificação em Java de um programa completo:

```
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

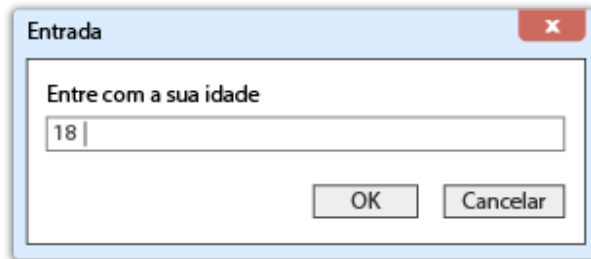
        //entrada de dados
        aux = JOptionPane.showInputDialog('Entre com a sua idade');
        //conversão de tipos
        idade = Integer.parseInt(aux);

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

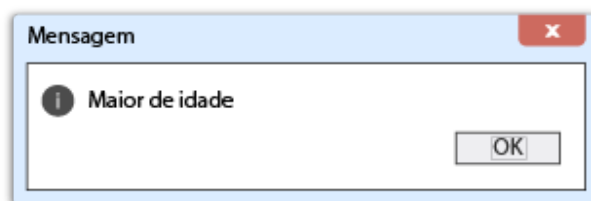
}//fim da classe
```

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:

A screenshot of a Windows-style dialog box titled "Entrada" (Input). It has a light blue header bar with a red close button (X) on the right. The main area is white and contains the text "Entre com a sua idade" (Enter your age). Below this text is a text input field containing the number "18". At the bottom right of the dialog are two buttons: "OK" and "Cancelar" (Cancel).

Teremos o seguinte retorno:

A screenshot of a Windows-style message box titled "Mensagem" (Message). It has a light blue header bar with a red close button (X) on the right. The main area is white and contains an information icon (i) followed by the text "Maior de idade" (Adult). At the bottom right of the dialog is a single button labeled "OK".

Note que se a condição for falsa (Não) nenhum comando é executado.

Estruturas de Decisão if-else

(Se... Senão... Fim-se)

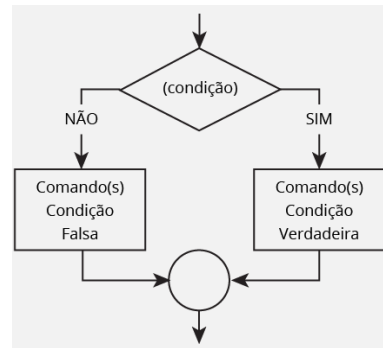
Você acabou de conhecer uma Estrutura de Decisão que realiza somente uma ação distinta caso o teste condicional seja verdadeiro. Contudo, em geral também costuma ser necessário que alguma ação seja realizada **caso o teste condicional seja falso**. Para isso, temos o comando if-else:

PSEUDOCÓDIGO

```

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

FLUXOGRAMA



JAVA

```

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

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...

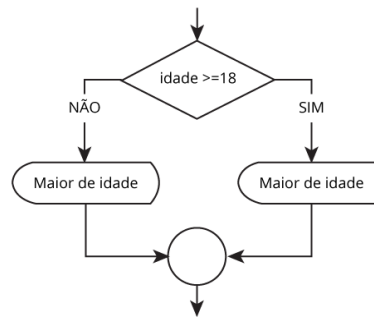
Considere o programa que analisava se uma pessoa era ou não maior de idade. Agora, porém, ele conta com comandos que serão executados se o teste condicional for falso.



PSEUDOCÓDIGO

```

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

FLUXOGRAMA**JAVA**

```

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

Perceba que o exemplo é praticamente idêntico ao caso anterior, porém, se o teste condicional falhar (caso o resultado seja falso, seguindo a resposta “não” do fluxograma), 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.

Agora, veja o 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

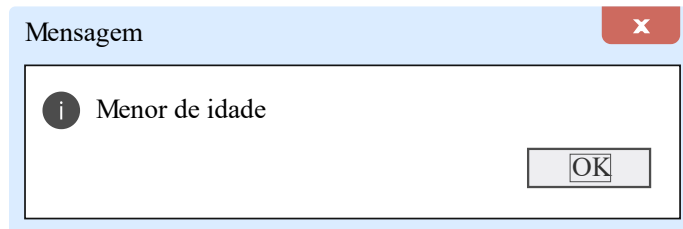
        //entrada de dados
        aux = JOptionPane.showInputDialog("Entre com a sua
idade");

        //conversão de tipos
        idade = Integer.parseInt(aux);

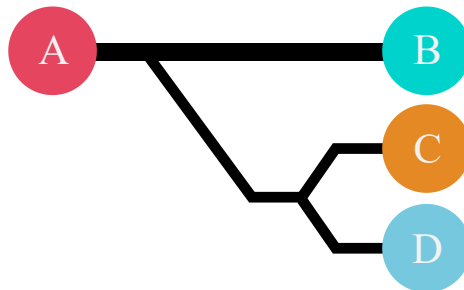
        //Decisão
        if (idade >=18) {
            //comandos se a condição for verdadeira
            JOptionPane.showMessageDialog(null, "Maior de Idade");
        } else {
            //comandos se a condição for falsa
            JOptionPane.showMessageDialog(null, "Menor de Idade");
        }
    }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

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

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



Estrutura de decisão aninhada



O que aconteceria se precisássemos de mais de duas alternativas para resolver um problema?

Talvez você tenha reparado que em uma Estrutura de Decisão podemos ter somente duas saídas: verdadeiro ou falso. Mas o que ocorre quando precisamos 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:

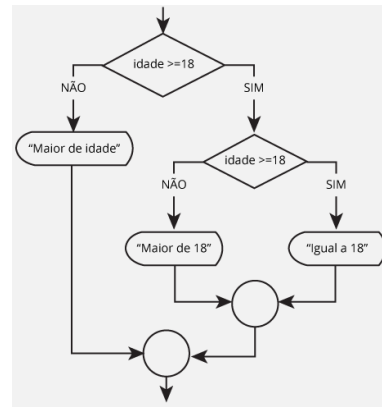
PSEUDOCÓDIGO

```

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

```

FLUXOGRAMA



JAVA

```

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

```

Note que ao executar a primeira tomada de decisão **SE** ($idade \geq 18$), 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. É isso que chamamos de Estrutura de Decisão Aninhada.

Agora, veja o programa completo codificado 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**.

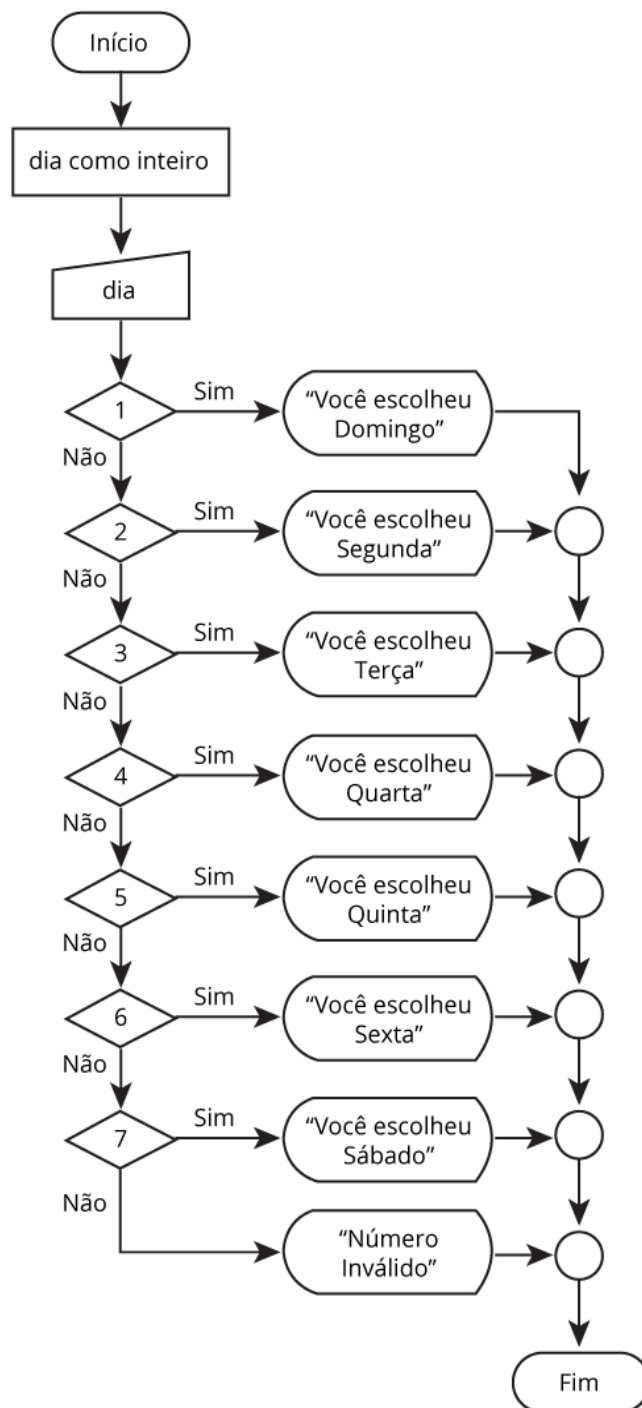
Qual seria o **tipo da variável** entrada?

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.

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.

```

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

```

Após observar os exemplos sobre Estruturas de Decisão Aninhadas, você deve estar se perguntando:

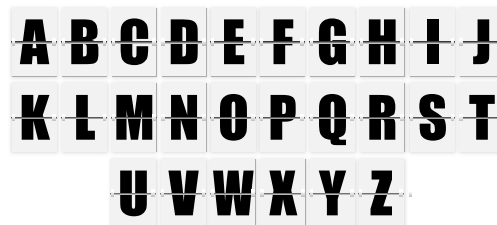
Existe alguma relação entre o **número de alternativas** e o **número necessário de comparações** a serem efetuadas?

Pare e reflita! 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

Até agora você viu como as Estruturas de Decisão se aplicam a situações que envolvem tipos numéricos, mas já pensou como poderíamos realizar comparações com uma **sequência de caracteres** em Java? Seria da mesma forma?

Na verdade, não. 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 abaixo:

```
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");

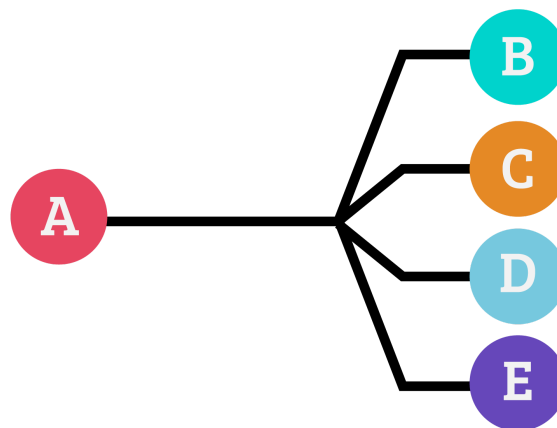
        //Processamento e saída
        if (nome.equals("Jose")) {
            JOptionPane.showMessageDialog(null, "O Nome
Digitado é Jose");
        } else {
            JOptionPane.showMessageDialog(null, "O nome
digitado foi " + nome);
        }
    }
}
```

Note que na linha 14 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)

Estrutura de Decisão caso... senão... fim_selecione

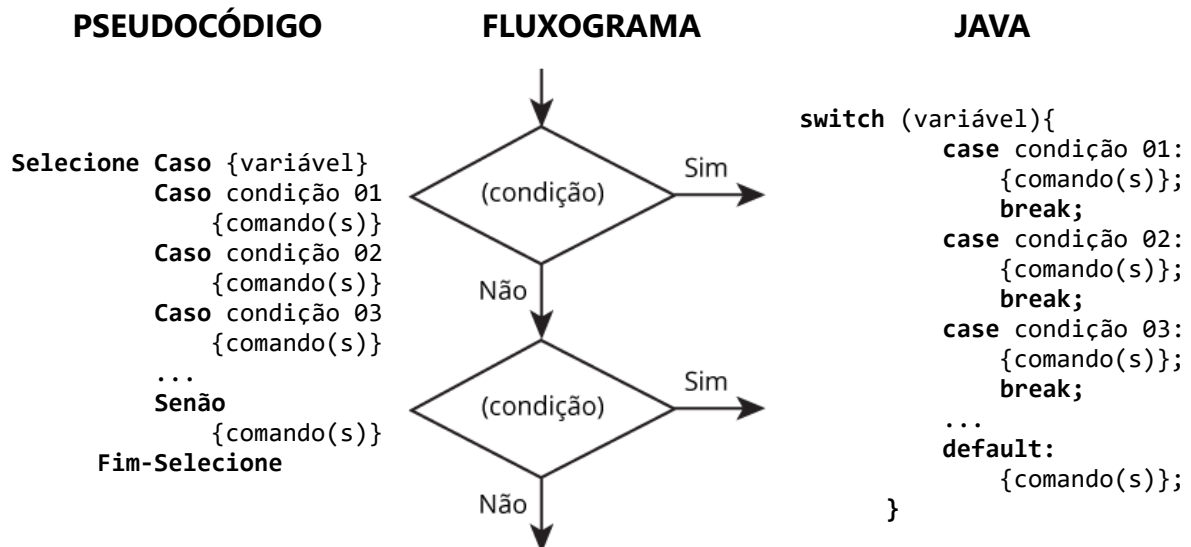


Já sabemos que a lógica de programação possui mecanismos que nos permitem tomar decisões dentro de um algoritmo. Sabemos também que esses mecanismos são denominados “Estruturas de Decisão”. A novidade é que essas estruturas não se restringem a apenas o “**se... senão...fim_se**”.

E se tivéssemos uma decisão a ser tomada entre **dez opções**? Será que o “**se...senão...fim_se**” seria a solução mais apropriada para essa situação? Será que existe alguma outra estrutura mais adequada para esse tipo de ocorrência?

Existe sim! Essa estrutura é chamada de **Selecione...caso...senão...fim_selecione** e sua função principal é **facilitar a escrita do algoritmo** quando se tem muitos caminhos a serem seguidos a partir de uma decisão. Assim como a estrutura “se...senão...fim_se”, é necessário saber quando e como utilizar o “selecione...caso...senão...fim_selecione”.

A estrutura “selecione...caso...senão...fim_selecione” do pseudocódigo corresponde à estrutura **switch-case** do Java. Elas permitem que a seleção correta seja feita a partir da comparação entre o valor do conteúdo da variável e uma lista definida durante a programação. Ao encontrar a correspondência correta, os respectivos comandos são executados e as demais opções ignoradas.



Se recapitularmos o último exemplo de Estrutura de Decisão Aninhada do tópico anterior, no qual o programa exibia o dia da semana de acordo com o número digitado pelo usuário, notamos que o código do programa fica relativamente **confuso** devido à grande quantidade de comandos de seleção (se). A tendência dessa confusão é aumentar conforme o número de comandos de decisão aninhados for crescendo. Isso, é claro, levando em conta que estamos realizando a comparação sempre com a mesma variável.

Para essas situações podemos utilizar a estrutura “selecione caso...senão...fim_selecione” do pseudocódigo ou a estrutura switch-case do Java.

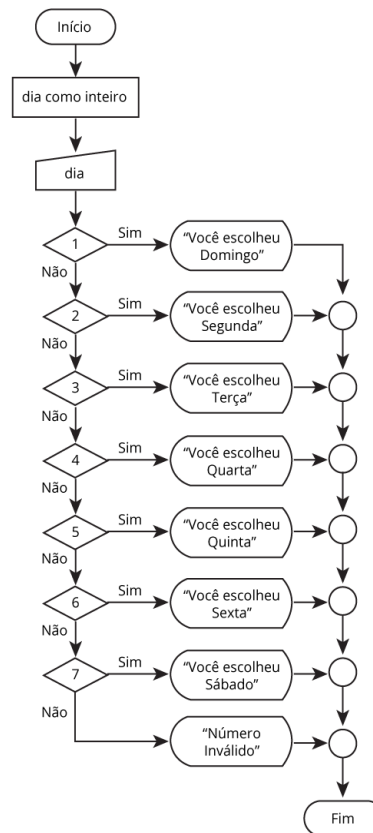
Tanto as sintaxes em pseudocódigo quanto em Java são semelhantes como temos a seguir:

PSEUDOCÓDIGO

```

Programa semana
Declare
dia como inteiro
Início
Escreva ("Digite um Número de 1 a 7")
Leia(dia)
Selecione Caso (dia)
Caso 1
Escreva ("Você Escolheu Domingo")
Caso 2
Escreva ("Você Escolheu Segunda")
Caso 3
Escreva ("Você Escolheu Terça")
Caso 4
Escreva ("Você Escolheu Quarta")
Caso 5
Escreva ("Você Escolheu Quinta")
Caso 6
Escreva ("Você Escolheu Sexta")
Caso 7
Escreva ("Você Escolheu Sábado")
Senão
Escreva("Número Inválido")
Fim-Seleção
Fim.

```

FLUXOGRAMA

Agora, veja o código em Java:

```

import javax.swing.JOptionPane;

public class caseDiaSemana {

    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

        switch (dia) {
            case 1:
                JOptionPane.showMessageDialog(null, "Você escolheu
Domingo");
                break;
            case 2:
                JOptionPane.showMessageDialog(null, "Você escolheu
Segunda");
                break;

```

```
        case 3:
            JOptionPane.showMessageDialog(null, "Você escolheu
Terça");
            break;
        case 4:
            JOptionPane.showMessageDialog(null, "Você escolheu
Quarta");
            break;
        case 5:
            JOptionPane.showMessageDialog(null, "Você escolheu
Quinta");
            break;
        case 6:
            JOptionPane.showMessageDialog(null, "Você escolheu
Sexta");
            break;
        case 7:
            JOptionPane.showMessageDialog(null, "Você escolheu
Sábado");
        default:
            JOptionPane.showMessageDialog(null, "Número
inválido");
            break;
    } // fim do switch-case
} // fim do método main
} // fim da classe
```

Você percebeu que esse código corresponde ao utilizado no exemplo anterior, utilizando a Estrutura de Decisão “se”? Ambos têm o mesmo efeito prático, mas com a estrutura “**caso**”, o código foi escrito de maneira mais elegante e o fluxograma permaneceu o mesmo sem alteração nenhuma.

Revisando as Estruturas de Decisão

Agora que você já estudou o conteúdo da unidade, assista ao vídeo do professor Sandro Valérius, preparado para o curso técnico de Informática, Módulo I - Agenda 13.

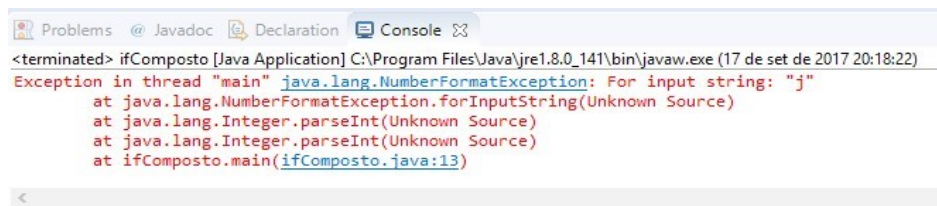
Informática - Módulo I - Agenda 13 - Estruturas d...



Ampliando Horizontes

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, 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)
```

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 grande problema realmente é o usuário ficar sem saber o que aconteceu. 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 também trata de outros tipos de erro, mas não é o escopo dessa explicação. A sintaxe é:

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

Vamos explicar com um exemplo:

```
import javax.swing.JOptionPane;

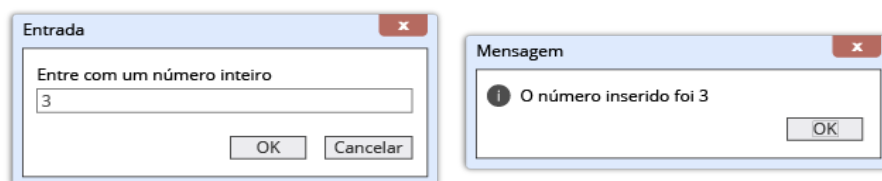
public class tryCatch {
```

```
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
```

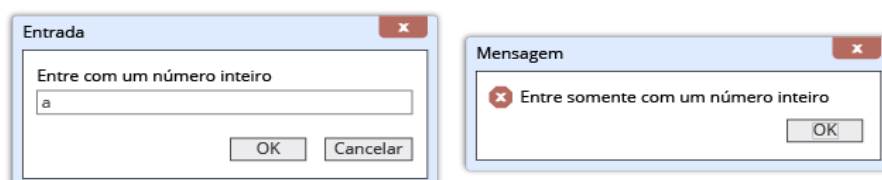
O try-catch funciona da seguinte forma:

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. Isso 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 ”, como nas figuras a seguir:



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



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: repare que quando você não trata o erro, aparece a seguinte mensagem para o usuário: **“Erro! Fonte de referência não encontrada”**. Bem diferente de quando você trata o erro com o comando try-catch e informa o usuário sobre o problema ocorrido para que ele possa atender às solicitações do programa. Analise as linhas de comando 16 e 17 para ver como o erro de entrada de dados pode ser tratado e qualquer dúvida contate o seu professor tutor!

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

O comando **JOptionPane.showMessageDialog (null, “Entre somente 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 formatação 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.

Mensagem de alerta

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

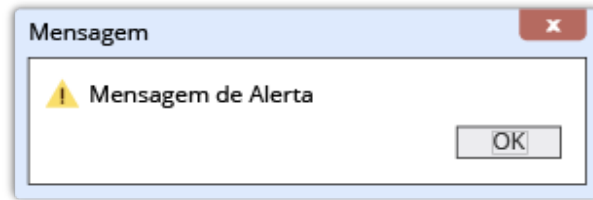
```
import javax.swing.JOptionPane;

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.



Você acabou de ver como pode modificar a exibição de uma caixa de diálogo de saída de dados. Explore um pouco na IDE Eclipse algumas outras personalizações dessa caixa de diálogo. Utilize a **função de auto completar comandos da IDE**: basta digitar o início de um comando e pressionar as teclas Control (CTRL) + barra de espaços.

Resumindo o Estudo

Nesta agenda, você viu que, assim como as decisões que você toma na sua vida impactam seu caminho, as decisões que softwares tomam a respeito dos dados que analisam podem determinar a resposta que ele dará ao usuário. Você conheceu diferentes comandos relacionados às Estruturas de Decisão e agora consegue desenvolver **programas cada vez mais complexos**, capazes de reagir aos dados inseridos pelos usuários.

Dominar as **Estruturas de Decisão** foi um grande avanço e na próxima agenda você terá seu primeiro contato com as **Estruturas de Repetição!**

Até lá!



