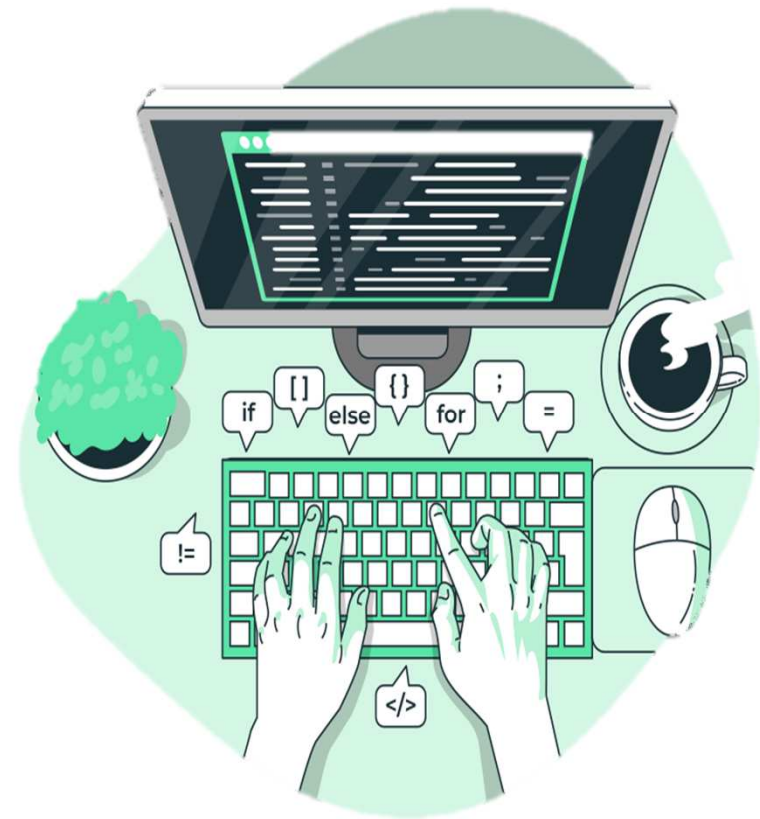


# CONSTANTES

Constante é uma referência que após definido um valor, este não poderá ser alterado durante o tempo de execução.

Na prática, o funcionamento é semelhante ao uso de variáveis, sendo a única diferença, que após definido um primeiro valor, este não poderá ser alterado.



# CONCEITO

Em programação, uma constante é a alocação de um espaço na memória para guardar um valor durante toda a execução de um programa ou então, durante a execução de um determinado Bloco de Instrução.

As constantes, são bastante semelhantes ao funcionamento e uso das variáveis, a diferença, é que não podemos alterar o seu valor em tempo de execução. Se, durante o desenvolvimento de um software, definirmos que uma constante deve conter o valor inteiro 5, está, em hipótese alguma, conseguira alterar o seu valor em tempo de execução.

Quando estamos iniciando na programação, é comum pensarmos que o uso de constantes ao invés de ajudar, atrapalha no desenvolvimento, o que muitas vezes pode ser uma verdade aparente. Assim, partimos do pressuposto que sempre lembraremos que determinada referência não poderá ter o seu valor alterado e então, optamos por trabalhar com uma variável, até porque, a mesma é mais flexível.

Esse tipo de pensamento, sob olhar da teoria está totalmente errado, e no olhar prático, estamos cometendo um erro gigantesco.

# CONCEITO

Temos que ter em mente, que o objetivo de uma constante, como o nome sugere, é assegurar que determinada informação não será alterada, em hipótese alguma. Trabalhar com constantes faz o nosso programa ser mais seguro e menos propenso a erros. Até porque, estamos garantindo que um valor constante não será alterado durante a execução da aplicação.

Em Java, definimos uma constante colocando a palavra final a frente da declaração, por exemplo:

```
final int NOME = 0;
```



# CONCEITO

Observe a instrução final à frente da declaração. Esta é a responsável por declarar que um determinado membro não poderá ter o seu valor alterado.

Em Java, existe uma peculiaridade quando trabalhamos com constantes. Nós podemos declarar uma constante sem inicializá-la. A partir do momento que um valor for atribuído, este não mais poderá ser alterado. Assim, temos a liberdade de definir o valor da constante de maneira dinâmica, o que facilita a programação, porém, pode ocasionar problemas.

Os nomes das constantes sempre devem estar em letra maiúscula, assim, conseguimos distinguir facilmente entre uma variável e uma constante. O Java não nos obriga a definir o nome das constantes com letra maiúscula, porém, essa é uma convenção da comunidade e devemos segui-la.

```
final double VELOCIDADE_SOM = 340,29; // m/s  
final double VELOCIDADE_LUZ = 299.792 458; // m/s  
final double PI = 3.14159
```

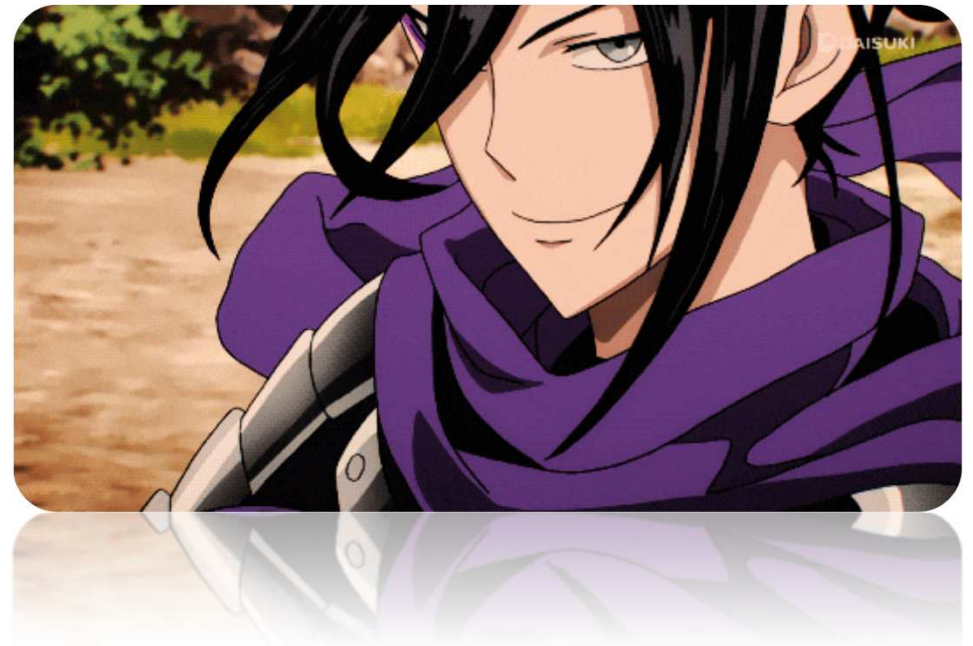
# EXEMPLO A SEGUIR

```
public class Aula0015 {  
  
    public static void main(String[] args) {  
        /*  
        * 1) os dados nunca serão alterados  
        * 2) uma constante tem o seu endereço protegido  
        * 3) uma constante nunca será desalocada  
        * 4) declaramos uma vez, utilizamos quantas vezes forem  
        necessárias  
        * */  
  
        int var = 10; //declaração de variável  
        final int constante = 10; //declaração de constantes  
  
        System.out.println(var );  
        System.out.println(constante );  
    }  
}
```

# CONCEITO

Continuando sobre as constantes, desenvolveremos um exemplo onde será realizado o cálculo da velocidade do som. Para isso, iremos declarar uma constante, ou seja, iremos definir a velocidade do som como um valor constante, até porque, em nosso programa, esse valor não será alterado.

Essa é mais uma das situações onde devemos utilizar valores constantes. Isso porque, se considerarmos a velocidade ideal, nós temos que a velocidade nunca se altera e assim, podemos definir esse valor em uma constante. Logo, todas as vezes em que precisarmos utilizar a velocidade do som em nosso código, temos que o valor estará numa constante e assim, não teremos a possibilidade de por exemplo, digitar a informação erroneamente.







## Seu Amigo

O GitHub Desktop é um aplicativo que permite que você interaja com o GitHub usando uma GUI em vez da linha de comando ou de um navegador web. GitHub Desktop incentiva você e sua equipe a colaborar usando as melhores práticas com Git e GitHub. Você pode usar GitHub Desktop para realizar a maioria dos comandos do Git a partir de seu computador com a confirmação visual das mudanças. Você pode fazer subir, extrair e clonar repositórios remotos com o GitHub Desktop e usar ferramentas colaborativas como atribuir commits e criar pull requests.

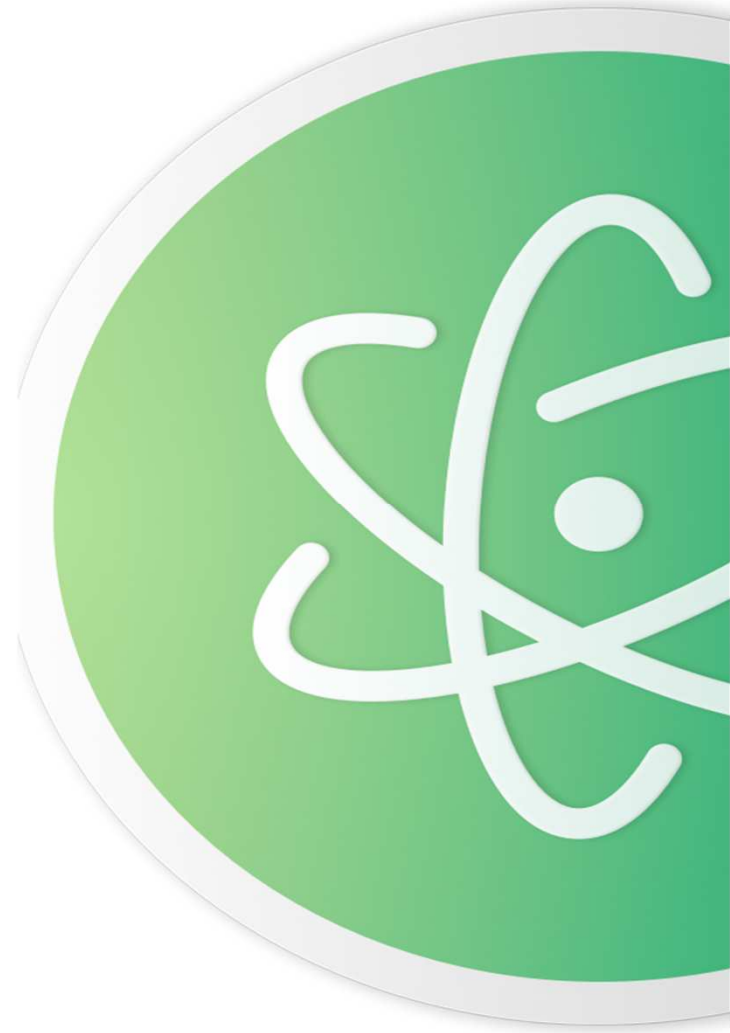




## Seu Amigo

<https://docs.github.com/pt/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop>

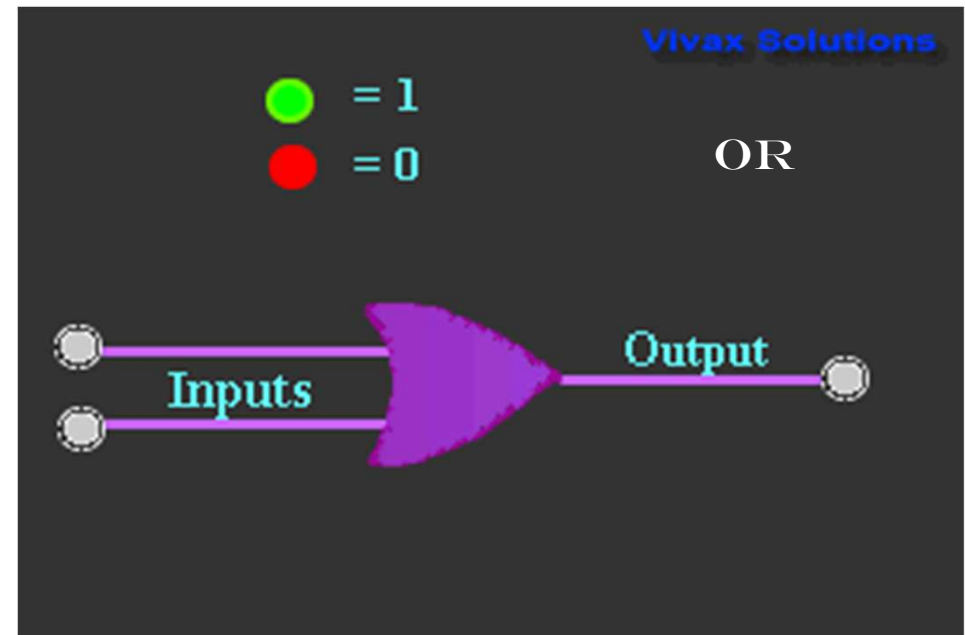
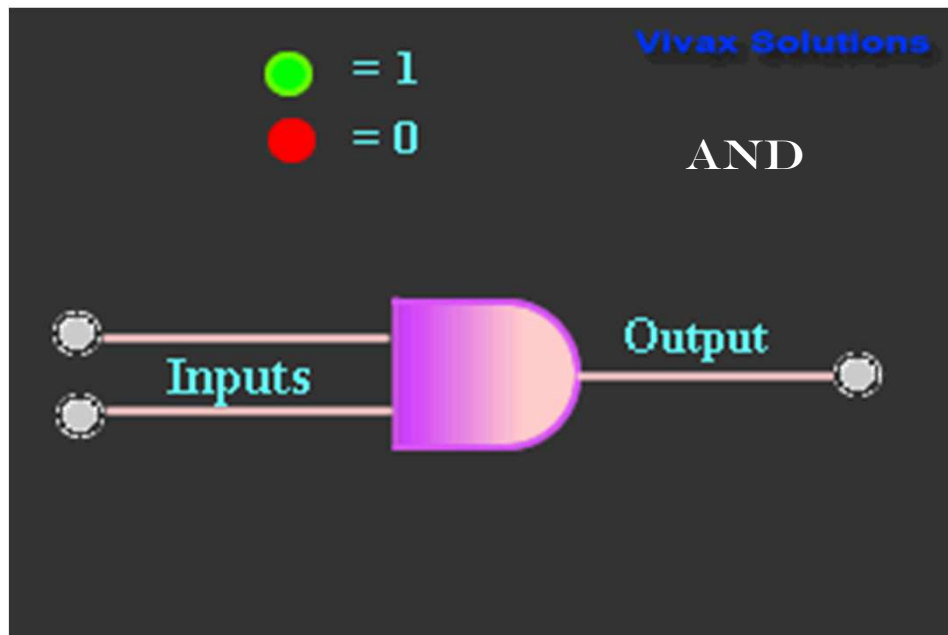
Atom é um editor de texto de código aberto disponível para as plataformas Linux, macOS e Microsoft Windows, desenvolvido pelo GitHub sob a licença MIT.



## EXEMPLO A SEGUIR

Quais os operadores lógicos do C++? Operadores lógicos constituem o conjunto de conectivos lógicos existentes na lógica fundamental. O C++ utiliza basicamente 2 conectivos lógicos emprestado da matemática, são eles:

1. Conectivo de conjunção -  $\wedge$  - E
2. Conectivo de disjunção -  $\vee$  - OU



## CONECTIVO ( E )

O conectivo E é utilizado nas expressões em que desejamos que todas as condições sejam verdadeiras, por exemplo:

```
var1 == 0 E var2 == 100
```

Temos que a primeira condição deve ser verdadeira (`var1 == 0`) como também a segunda condição (`var2 == 100`). Caso uma das condições não seja verdadeiro, o valor lógico ser falso. Assim, o conectivo E exige que todas as condições sejam verdadeiras para que o valor lógico da expressão seja verdadeiro.a

# CONECTIVO ( E )

Ideia por trás do operador "E"


Você pode obter uma habilitação de motorista se:


- For aprovado no exame psicotécnico,
- E**
- For aprovado no exame de legislação,
- E**
- For aprovado no exame de direção


**Todas condições  
devem ser  
verdadeiras!**

Exemplos de expressões lógicas

(suponha x igual a 5)

$X \leq 20 \ \&\& \ X == 10$  Resultado: F  


$X > 0 \ \&\& \ X != 3$  Resultado: V  


$X \leq 20 \ \&\& \ X == 10 \ \&\& \ X != 3$  Resultado: F  


## CONECTIVO ( OU )

O conectivo OU diz que uma das expressões deve ser verdadeira, por exemplo:

```
var1 == 0 OU var2 == 100
```

Assim, temos que se a primeira condição for verdadeira (`var1 == 0`) ou então, a segunda condição (`var2 == 100`), então o valor de toda a expressão será verdadeiro.

# CONECTIVO ( OU )

## Exemplos de expressões lógicas

(suponha x igual a 5)

$X == 10$  ||  $X <= 20$       Resultado: V  
F                      V

$X > 0$  ||  $X != 3$       Resultado: V  
V                      V

$X <= 0$  ||  $X != 3$  ||  $X != 5$       Resultado: V  
F                      V                      F

## Tabela verdade do operador "OU"

A	B	A    B
F	F	F
F	V	V
V	F	V
V	V	V

# Operadores Comparativos

C, C++,  
Java, C#



Operador	Significado
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
==	igual
!=	diferente

## Exemplos de expressões comparativas

(suponha x igual a 5)

$X > 0$

Resultado: V

$X == 3$

Resultado: F

$10 <= 30$

Resultado: V

$X != 2$

Resultado: V



# Estrutura switch-case

Quando se tem várias opções de fluxo a serem tratadas com base no valor de uma variável, ao invés de várias estruturas if-else encadeadas, alguns preferem utilizar a estrutura switch-case.

## Problema exemplo

Fazer um programa para ler um valor inteiro de 1 a 7 representando um dia da semana (sendo 1=domingo, 2=segunda, e assim por diante).  
Escrever na tela o dia da semana correspondente, conforme exemplos.

Entrada	Saída
1	Dia da semana: domingo

Entrada	Saída
4	Dia da semana: quarta

Entrada	Saída
9	Dia da semana: valor inválido

# Estrutura switch-case

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        String dia;

        if (x == 1) {
            dia = "domingo";
        }
        else if (x == 2) {
            dia = "segunda";
        }
        else if (x == 3) {
            dia = "terca";
        }
        else if (x == 4) {
            dia = "quarta";
        }
        else if (x == 5) {
            dia = "quinta";
        }
        else if (x == 6) {
            dia = "sexta";
        }
        else if (x == 7) {
            dia = "sabado";
        }
        else {
            dia = "valor invalido";
        }

        System.out.println("Dia da semana: " + dia);
        sc.close();
    }
}
```

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        String dia;

        switch (x) {
            case 1:
                dia = "domingo";
                break;
            case 2:
                dia = "segunda";
                break;
            case 3:
                dia = "terca";
                break;
            case 4:
                dia = "quarta";
                break;
            case 5:
                dia = "quinta";
                break;
            case 6:
                dia = "sexta";
                break;
            case 7:
                dia = "sabado";
                break;
            default:
                dia = "valor invalido";
                break;
        }

        System.out.println("Dia da semana: " + dia);
        sc.close();
    }
}
```

# Estrutura switch-case

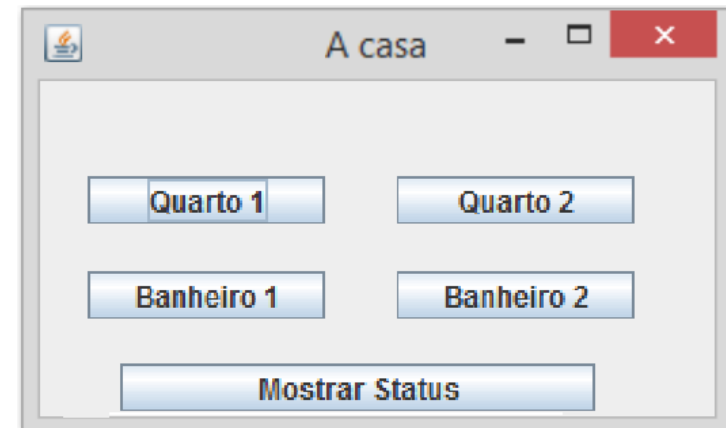
## Sintaxe do switch-case

```
switch ( expressão ) {  
  case valor1:  
    comando1  
    comando2  
    break;  
  case valor2:  
    comando3  
    comando4  
    break;  
  default:  
    comando5  
    comando6  
    break;  
}
```

# JFrame

A classe JFrame nos possibilita a criar uma tela para nossa aplicação . Dentro desta tela podemos incrementar outros componentes como:

- Caixa de texto
  - Botões
  - Radio botões
  - Texto
  - Imagens
  - Área de texto
- e assim por diante



# JFrame

**import javax.swing.JFrame;** //Parte que vai receber os Botões

**import java.awt.event.ActionEvent;** //Parte que vai adicionar os Botões

**import java.awt.event.ActionListener;** //Parte que vai ligar o Botão

**import javax.swing.JButton;** //Cria o Botão

**import java.awt.GridLayout;** //Cria O grid para os botões não ficarem em cima do outro

## Adicionando um evento

```
public class Casa extends JFrame{
    public static void main(String[] args) {
        JFrame casa = new JFrame("A Casa");
        JButton jbQuarto1 = new JButton("Quarto 1");
        jbQuarto1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
            }
        });
        casa.add(jbQuarto1);
        casa.setLayout(new GridLayout());
        casa.setVisible(true);
    }
}
```

- *Faça os outros eventos*

# JFrame

