

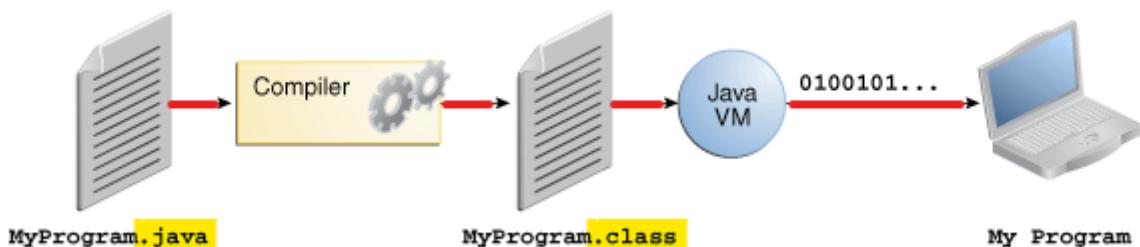
## Instalando Java

- Download do kit developer [Java JDK](#)
- Download da biblioteca gráfica [JavaFX Scene Builder](#)
- Download da IDE Apache [NetBeans](#)

Explicação do conteúdo das pastas de um projeto NetBeans:

- **src**: (source) contém os pacotes, as classes, as imagens e o código-fonte (.java)
- **build**: guarda os bytecodes com extensão **.class**, eles são os arquivos que passaram pelo compilador
- **dist**: guarda os arquivos com extensão **.jar**, que são executados pela JRE, destinados para distribuição e compartilhamento do programa
- **nbproject**: guarda as **configurações** do seu NetBeans, definindo que esse conjunto de pastas são um projeto para o framework Netbeans

Para salvar/construir um arquivo executável e distribuir o projeto no NetBeans, clique em Builder (martelo) e depois abra o explorer → vá na pasta “meus documentos” → “Netbeans Projects” e copie a pasta inteira(dist). A extensão é **.jar** e é o **bytecode** que a JRE executa



## Criando o primeiro projeto

Dentro do NetBeans, na aba Arquivos, crie **novo projeto** → categorias: java, projetos: aplicação java → edite o nome do projeto, e **desmarque a caixa** “criar classe principal”, botão finalizar → na guia à esquerda que se abriu, clique em “pacotes código-fonte”, botão direito, criar **novo pacote** (esses pacotes contêm as classes) → na janela que se abre, edite o nome da classe (use **CamelCase**: inicie palavras com letras minúsculas!) → repita o passo anterior se quiser criar um pacote para guardar imagens → novamente na guia à esquerda, clique no pacote criado, botão direito, criar **novo Form JFrame** → edite o nome da classe usando CamelCase

# Aula 01

➤ **Resumo da história do Java:** (café forte das ilhas de Java - Jamaica) James Gosling e sua equipe trabalhavam na *Sun*, no projeto *Star Seven*, que tinha o objetivo de interligar várias interfaces, fazendo com que diversos dispositivos diferentes conversassem entre si. Esse projeto foi engavetado mas foi construído na *Linguagem Oak* (em homenagem ao carvalho em frente à sala da equipe). Tim Bernes Lee aparece com o *HTML* para facilitar o uso de navegadores, incentivando a *Sun* a dar origem ao navegador *WebRunner*. Como o nome “*Oak*” já havia sido patenteado, foi preciso rebatizar a linguagem para um novo nome: *Java*. Surgindo assim, o *Browser HotJava*. Em 2009 a *Oracle* comprou a *Sun*.

---

- **Linguagem de alto nível** é uma linguagem mais próxima da compreensão do usuário por ser mais intuitiva, amigável e fácil de aprender. Essas linguagens abstraem conceitos voltados para a máquina e sintetizam comandos (elas estão longe das microinstruções de máquina)
- O computador não entende o código-fonte gerado por linguagens de alto nível, é preciso que esse código seja transposto para um **compilador**.

**Compilador** é um software que traduz o código gerado por uma linguagem de alto nível para um **código-objeto** (sinônimo de **código executável**), ou seja, um programa equivalente, contendo microinstruções de máquina, comprehensíveis a um processador e responsáveis por executar esse programa. O problema é que nas linguagens que precisam de compilador, é necessário um compilador para cada plataforma: Windows, Mac, Linux, Android, IOS, etc.

O processo de compilação é usar compilador e o montador. O compilador interpreta todos os comandos, verifica se há erros/inconsistências no código fonte e depois traduz isso tudo para o montador gerar o código-executável. O arquivo produzido pelo compilador tem a extensão **.class**

Veja a diferença entre compilador e interpretador e o processo compilação:

- O **compilador** executa todo o código de uma vez, fazendo com que um código compilado **ocupe mais memória** porém execute em **menos tempo**.
- Já o **interpretador** traduz uma linha de cada vez o que faz com que **ocupe menos memória** porém **leva mais tempo**.

CÓDIGO FONTE → **compilador** → CÓDIGO OBJETO → **assembler/linker/montador** → CÓDIGO EXECUTÁVEL

ou

CÓDIGO FONTE → **interpretador** (faz a tradução direta para) → EXECUTÁVEL

## Aula 02

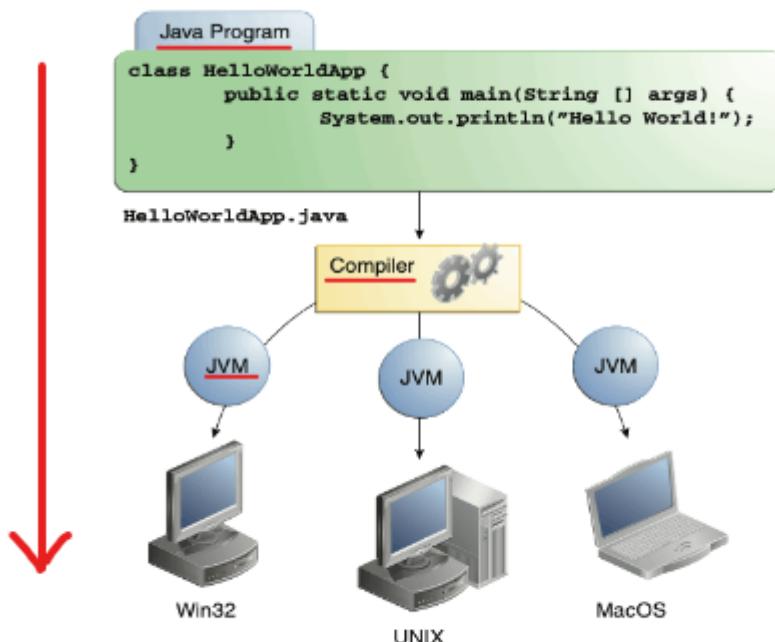
A diferença da linguagem Java é que seu compilador, o **JavaC**, não gera um código executável como os demais compiladores, ele **gera um ByteCode**, que é comum para qualquer plataforma.  
Então o JavaC transforma o código-fonte em Bytecode.

A **JVM** - Java Virtual Machine - recebe o ByteCode, interpreta-o e gera o código necessário para a máquina entender. Existem JVMs para cada plataforma, o que torna essa linguagem multiplataforma.

Então a qualidade principal do Java é **WORA** - Write Once, Run Anywhere.

Resumindo:

CÓDIGO FONTE → JAVAC → JVM → Windows, Mac, Android, Linux



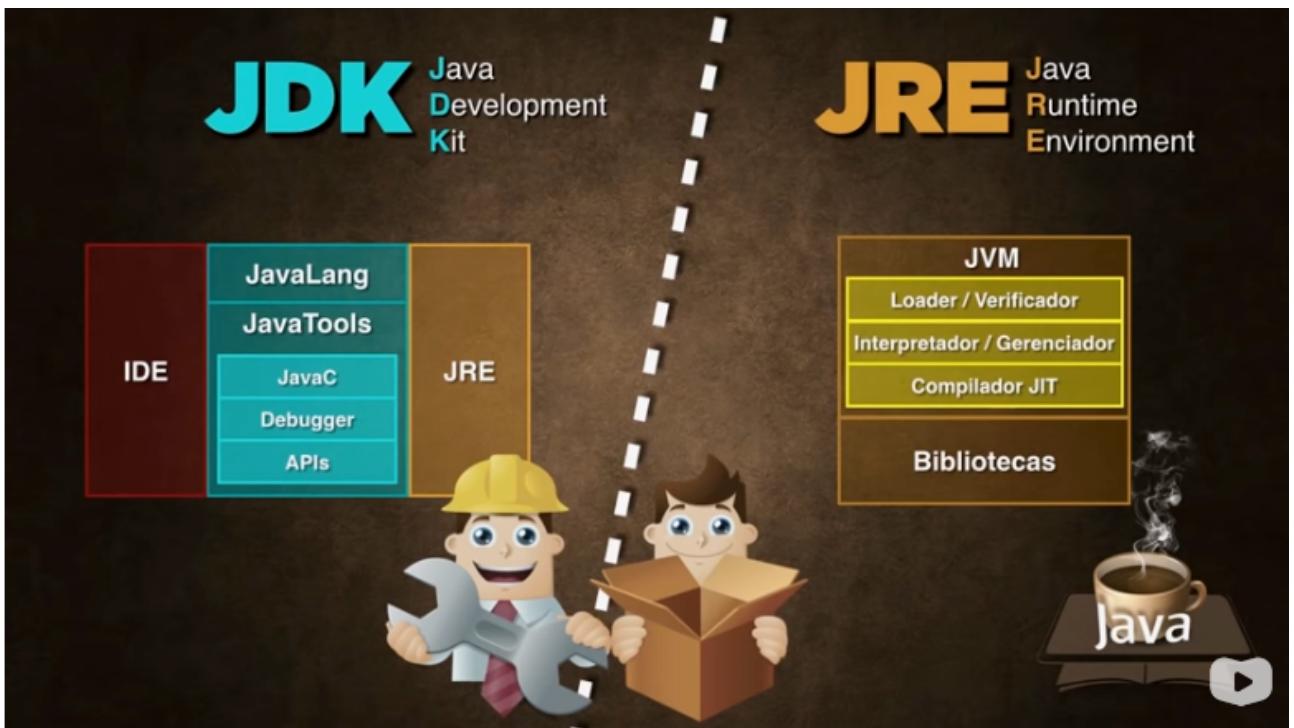
Through the Java VM, the same application is capable of running on multiple platforms.

**JRE** - Java Runtime Environment: Ambiente de execução criado para aqueles que querem apenas usar o JAVA. É dividido em 2 partes:

- Bibliotecas
- Loader / Verificador / gerenciador de memória / compilador JIT (aumenta a performance)

**JDK** - Java Developer Kit: Kit de desenvolvimento criado para aqueles que querem desenvolver em JAVA. É dividido em 3 partes:

- JRE
- IDE (com API's, Java Lang, JavaTools [JavaC, Debugger])
- JVM



Uma API é criada quando uma empresa de software tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço. Elas funcionam através da comunicação de diversos códigos, definindo comportamentos específicos de determinado objeto em uma interface. A API liga as diversas funções em um site de maneira que possa ser utilizada em outras aplicações. Sistemas de pagamento online são um bom exemplo, assim como o Google Maps.

**IDE** - Integrated Development Environment. São ferramentas, ambientes para desenvolver programas. Exemplos de IDE's do Java:

- **NetBeans** - Produzido pela Oracle, passada para a Apache
- **Eclipse**
- **IntelliJ**

## Vantagens de uma IDE



## Aula 03

Existem 3 distribuições Java:

-SE: Standard Edition

-EE: Enterprise Edition (contém acesso remoto, acesso a grande BDs, etc)

-ME: Micro Edition (Smartphones, Wearables, etc)

## Aula 04

Quase todo programa Java é composto de pacotes (packages). Pacote é composto por **classes**, classe é composta por **métodos** (funções ou procedimentos), método é composto por **atributos** (dados)

Uma classe é um bloco separado pelas chaves { }

A presença da palavra void indica que é um método sem retorno (procedimento)

Todo comando/instrução em Java deve terminar com ponto-e-vírgula(;)

Pular/quebrar linha pode ser feito de 2 formas:

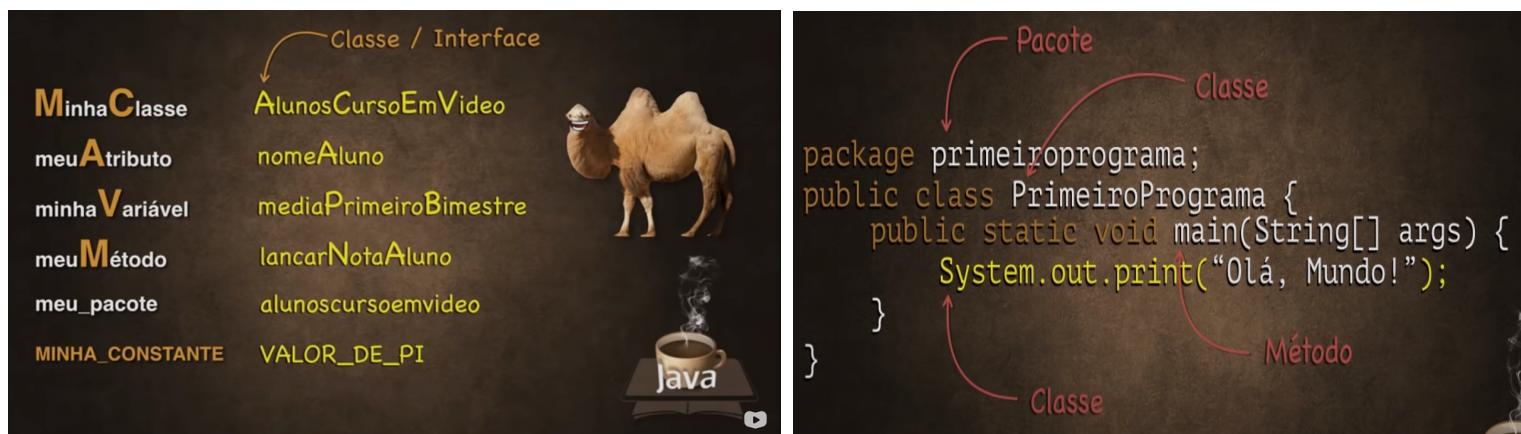
- Adicionando Ln ao final do comando de imprimir: System.out.println()
- Ou inserindo \n dentro de uma string: “ Você é maior de idade\n ”

Imprimir variáveis (especialmente variáveis float) pode ser feito de 2 formas:

- Usando **função format**: `System.out.printf("Seu peso é %.2f ", peso);`
- Usando **concatenação**: `System.out.print("Sua média foi " + peso);`

**Técnica do CamelCase:** Java é uma linguagem fortemente tipada, o que significa que ela diferencia letras maiúsculas de minúsculas (**Case Sensitive**)

- **Package**: tudo minúsculo e sem espaços (podendo usar o underline)
- **Class**: primeiras letras em Maiúsculo. Exemplo 'MinhaClasse'
- Atributo, variável, parâmetro ou **método**: A 1º letra em minúsculo



As variáveis dentro de uma classe 'private class' não conversam com as variáveis de uma classe 'public class'

O exemplo a seguir mostra um **método principal** (main), que está dentro da **classe** 'PrimeiroPrograma' e que faz parte do **pacote** 'primeiroprograma'. Esse método main é o primeiro a ser chamado e executado pela JVM:

```
package primeiroprograma;
public class PrimeiroPrograma {
    public static void main (String [] args) {
    }
}
```

**Alguns Métodos importantes:**

**System.out.println(" ");** //podendo escrever apenas sout e apertar TAB

**public static void main (String [ ] args);** // podendo escrever apenas psvm e apertar TAB, ele também funciona sem parâmetros

**Scanner teclado = new Scanner(System.in);** //cria um novo objeto `teclado` , (pertencente à classe Scanner que, por sua vez, pertence à biblioteca java.util) e esse objeto vai monitorar a entrada de dados do sistema.

**String nome = teclado.nextLine();** //cria uma string `nome` , que recebe o retorno de uma função `nextLine` , do objeto `teclado` para ler a entrada de caracteres

**float nota = blahblah.nextFloat();** //cria uma variável `nota` do tipo float para ler a entrada de dados float(decimal)

**System.out.printf("%.3f",idade);** //formata a quantidade de casas decimais do float que será impresso

**.setFont(new Font("Arial Black", Font.PLAIN, 22));** altera a fonte

**int ano = Calendar.getInstance().get(Calendar.YEAR);** //Através da classe `Calendar` , usa função dentro de função para entregar o ano atual

**System.getProperty("user.name");** exibe o nome do usuário  
**System.getProperty("os.name");** exibe o sistema operacional

**labelBerimbau.setText(berimbau);** copia/coloca o texto da variável `berimbau` dentro do objeto label `Berimbau` , mudando seu conteúdo  
**spinnerBerimbau.getValue();** // pega o valor contido no spinner `Berimbau`

## Aula 05

Por padrão, o Java não vem com a maioria das bibliotecas, apenas com a **JavaLang** que contém as instruções essenciais.

**Bibliotecas** são um conjunto de subprogramas (classes com funções).

O comando **import**: importa novas bibliotecas. Exemplo de algumas:

- java.applet: para criar aplicativos
- java.util: utilitários como monitoramento de teclado, entrada de dados
- java.math: para funções matemáticas
- java.net: para redes
- jacax.sound
- javax.media
- javax.swing
- javafx.fxml

A biblioteca **Swing** (evolução da antiga AWT) permite criar interface gráfica para ambientes de janelas (Windows, Linux), mas não suporta Android

Já o **JavaFx** é uma nova biblioteca gráfica, uma evolução da Swing. E por ser uma plataforma adicional de software, é preciso fazer o download do JavaFX SceneBuilder (muito utilizado para construir telas pois não só cria Apps de janela mas também para smartphones, browsers, IoT....)

No JavaForms você cria o ambiente (janela) e depois clica com o botão direito no componente a ser codificado.

Abaixo segue o código para configurar a mensagem de um label a partir de um button:

(Em outras palavras, quando apertar o botão aparecerá a mensagem na tela)  
lblMensagem.setText("OláMundo");  
nome do label → comando → mensagem

- extends: está relacionado ao conceito de herança
- private: tornar coisas privadas ao objeto, ou seja, encapsulamento
- evento: é um método, em resposta a alguma coisa
- implement: significa que está fazendo uma interface

Abaixo segue um código usando a biblioteca Swing:

```
public class TelaSwing extends javax.swing.JFrame {  
    private javax.swing.JButton btnClick;           //encapsulamento e  
    private javax.swing.JLabel lblMensagem;        // declaração de controles  
    private void btnClickActionPerformed...();      //evento  
    lblMensagem.setText("Olá Mundo!");
```

---

#### Explicando o código:

---

Na 1º linha: A classe pública TelaSwing tem como herança o Jframe (da biblioteca Swing), ou seja, tudo o que o Jframe tiver já construído vai passar para a tela do Swing.

Na 2º linha: Ainda dentro da classe, temos as especificações de cada controle/comando (btnClick é o nome de um JButton que é do tipo Swing).

Na 3º linha: temos o código de método / evento

Na 4º linha: temos o método setText() do objeto que vai poder modificar o texto que, por sua vez, está dentro do label.

---

Ao criar um programa com tela (Jform), utilizando as bibliotecas Swing ou JavaFx, não esqueça de desmarcar a textBox de criar classe principal.

Método Construtor **initComponents()** é uma rotina que é executada assim que o objeto é criado (aparece na tela). Ele tem o mesmo nome da classe.

Para trabalhar com pilhas é preciso carregar a classe stack, dentro da java.util

A palavra **new** cria um objeto e, para isso, deve haver uma classe referenciando ele.

## Aula 06

### Tipos Primitivos de Manipulação de Dados:

comentários: utilizar **//** ou **/\*comentário com mais de uma linha\*/**

Em relação às **variáveis**:

- Se todas as letras forem maiúsculas elas identificam constants.
- Seus nomes não começam com número ou ter sinais alfanuméricos!
- As variáveis são declaradas no rodapé do programa.
- Não declare variáveis dentro de estruturas de repetição.
- Existem 4 tipos de variáveis:  
int, float, boolean e char: (inteiro(**d**), real(**f**), lógico e caractere).

**Não existe o tipo String!** Mas sim a classe invólucro String. Então cria-se um objeto a partir da classe String, lembrando que objeto inicia com Maiúscula. Os tipos Inteiro ou char não podem receber string mas uma string pode receber inteiros ou char.

Para o Java, qualquer texto entre aspas duplas é uma instância da classe String. Exemplo:

```
String contagem = " "
String texto = "Qualquer texto entre aspas é uma String"
```

### Existem 3 formas de declarar variáveis no Java:

#### ➤ Por atribuição de valor

- **int idade = 3;**
- **float salario = 1825.54f;** // não esquecer do **f** no final
- **char letra = 'J';** // são aspas 'SIMPLES'
- **boolean casado = false;** // o tipo lógico é conhecido como booleano

```
int idade = 3;
float sal = 1825.54f;
char letra = 'G';
boolean casado = false;
```

➤ Por Typecast

- int idade = (int) 3;
- int salario = (float) 1825.54; // note que não precisou do f
- char letra = (char) 'J';
- boolean casado = (boolean) false;

```
int idade = (int) 3;
float sal = (float) 1825.54;
char letra = (char) 'G';
boolean casado = (boolean) false;
```

typecast

➤ Por Wrapper Class (classe invólucro)

- Integer idade = new Integer(3); //Perceba que o tipo inicia com maiúscula
- Float salario = new Float(1825.54); //Note que os valores estão entre parênteses.
- Character letra = new Character('J');
- Boolean casado = new Boolean(false);

```
Integer idade = new Integer(3);
Float sal = new Float(1825.54);
Character letra = new Character('G');
Boolean casado = new Boolean(false);
```

Wrapper Class

E para justamente para economizar memória é que existem muitos tipos primitivos:

Família	Tipo Primitivo	Classe Invólucro	Tamanho	Exemplo
Lógico	boolean	Boolean	1 bit	true
Literais	char	Character	1 byte	'A'
	-	String	1 byte/cada	"JAVA"
Inteiros	byte	Byte	1 byte	127
	short	Short	2 bytes	32 767
	int	Integer	4 bytes	2 147 483
	long	Long	8 bytes	$2^{63}$
Reais	float	Float	4 bytes	$3.4e^{+38}$
	double	Double	8 bytes	$1.8e^{+308}$

Cuidado! O Java trata tipos primitivos da maneira primitiva. Analise o código:

```
int n1 = 2, n2 = 5;
double resultado = n2/n1;
System.out.print(resultado);
```

O Java vai dividir um inteiro por outro e retornar o valor **inteiro** dele! Sendo assim, o programa vai retornar o valor 2.0 e não 2.5!

### Formas para Saída de Dados:

- `System.out.print ("sua nota é" + nota); //atenção para a concatenação (+)!`
- `System.out.println ("sua nota é" + nota); // \n é pra pular linha`
- `System.out.printf ("A nota de %s é %.2f \n" , nome, nota);`
- `System.out.format ("A nota de %s é %.2f \n" , nome, nota); //a sintaxe é exatamente a mesma. Perceba o f no fim de printf: assim não é necessário o (+) de concatenação, porém não esqueça do % que, por sua vez, indica uma inserção de uma variável dentro da string (no exemplo tem uma string e um float). O .2 determina a quantidade de casas decimais do float. A contrabarra \n pula linha. E não esqueça das vírgulas entre as variáveis.`

### Formas para Entrada de Dados:

É necessário importar uma classe `Scanner` da biblioteca `java.util` pois, por padrão, o Java não vem com nenhum comando de entrada de dados.  
Para isso, utilize o comando: `import java.util.Scanner;`

Pra ativar essa classe crie um objeto: `Scanner teclado = new Scanner(System.in);` //tem de estar dentro da função principal (psvm)

Uma classe representa um objeto. Se quiser usar o objeto ‘teclado’ para ler um número inteiro, escreva:

```
int idade = teclado.nextInt();
```

Então `nextInt()` é um método, do objeto teclado, para ler um número inteiro. Podemos usar o método `nextFloat()` para ler números reais, o método `nextBoolean` para lógicos, `nextDouble`, etc. Exemplo:

```
Int pernas = teclado.nextInt(); ou String pernas = teclado.nextLine();
```

No exemplo acima, para ler uma String, foi usado a classe invólucro String com o método `nextLine()`

### Incompatibilidades entre números e Strings e conversão:

A linguagem Java é fortemente tipada então existe uma grande incompatibilidade entre valores numéricos e valores String. Eles não se conversam. A maneira de solucionar isso é utilizando classe invólucro (wrapper). Segue os comandos:

- converter um **inteiro** para **String**:

```
int idade = 30;  
String valor = Integer.toString(idade);
```

- converter uma **String** para **inteiro**):

```
int valor = Integer.parseInt(spinnerValor.getValue().toString());
```

- (converter uma **String** para **real**):

```
float valor = Float.parseFloat(spinnerValor.getValue().toString());
```

- converter uma **String** para **double**):

```
double valor = Double.parseDouble(spinnerValor.getValue().toString());
```

- converter uma **String** para **boolean**):

```
boolean valor = Boolean.parseBoolean(spinnerValor.getValue().toString());
```

Em resumo:

- **Integer, Float, Double, Boolean** (começando com Maiúsculo) são classes que contém vários métodos como **toString()**, **parseInt()**, **parseFloat()**, **parseDouble()**, **parseBoolean()**, etc...
- **toString()** é um método que converte um objeto em string
- **parseInt()** é um método que converte um objeto em inteiros

E para **converter inteiros em float** ou vice-versa? É só criar uma Typecast

```
float restloat = (Num % Den);
```

```
int restint = (int) restloat; //cria uma nova variável `restint`-int para
receber o valor da variável `restloat`-float, através da forma Typecast
```

### Métodos assessores:

Método **Getter()**: pega um valor de um componente do JForm. Exemplo:  
`get.Text()` pega o texto de uma TextBox.

Método **Setter()**: coloca um valor, então o **set.Text()** muda o texto do TextBox  
`labelBerimbau.setText(berimbau);`

## Aula 07 e 08

Classe Math			
<b>PI</b>	Constante $\pi$	<code>Math.PI</code>	3.1415...
<b>pow</b>	Exponenciação	<code>Math.pow(5,2)</code>	25
<b>sqrt</b>	Raiz Quadrada	<code>Math.sqrt(25)</code>	5
<b>cbrt</b>	Raiz Cúbica	<code>Math.cbrt(27)</code>	3



O Java não vem com um operador de exponenciação, para isso, é preciso importar a **classe Math**, que contém métodos/funções próprias à matemática. Lembrando que classe sempre começa com letra Maiúscula. A classe **Math.pow** usa o tipo **Double**.

# Arredondamentos

<b>abs</b>	Valor Absoluto	Math.abs(-10)	10	
<b>floor</b>	Arredonda para Baixo	Math.floor(3.9)	3	
<b>ceil</b>	Arredonda para Cima	Math.ceil(4.2)	5	
<b>round</b>	Arredonda Aritmeticamente	Math.round(5.6)	6	

- ◆ **Valor Absoluto (abs)** é o mesmo que |módulo|: -10 vira 10
- ◆ **Truncagem (floor)** é o arredondamento para baixo: 3.9 vira 3
- ◆ **Ceil** é o contrário, arredondando para cima: 4.4 vira 5
- ◆ **Arredondamento (round)** Aritmético e tradicional: 4.4 vira 4 e 4.6 vira 5

O separador decimal é o ponto (não é a vírgula)!

## Método para gerar valores aleatórios:

```
double aleatorio = Math.random() //gera um número entre 0 e 1
```

A seguinte fórmula gera valores entre 15 e 50:

```
int numero = (int) (aleatorio * (50 - 15) + 15); ]
```

## Operadores:

### ➤ Operadores Relacionais:

>	Maior que	5 > 2	true
<	Menor que	4 < 1	false
>=	Maior ou igual a	8 >= 3	true
<=	Menor ou igual a	6 <= 6	true
==	Igual a	9 == 8	false
!=	Diferente de	4 != 5	true

Em Java, o sinal (**=**) é para atribuição de valores (assim como o “**→**” é para a linguagem C). Então não confundir com **Igualdade** (**==**)

Cuidado ao comparar usando o operador de igual (**==**), pode haver diferença entre as estruturas!

- O operador **==** verifica se ambos os objetos apontam para a mesma localização de memória.
- Já o método **equals()** faz a comparação de valores nos objetos.

Então, para comparar apenas o que está dentro da variável/objeto, utilize o método **Equals()**. Exemplo:

```
resultado = (nome1.equals(nome3))?"igual":"diferente";  
em vez de: System.out.println (parte 3 == parte 4);
```

➤ Operadores Unários:

++	Incremento	a ++	a = a + 1
--	Decremento	a --	a = a - 1

Faz diferença a posição dos incrementos ou decrementos (++ ou --)

**Pré-incremento:** é antes da variável (Ex.: ++5): vai adicionar +1 ao 5, passando a valer 6 imediatamente

**Pós-incremento:** é depois da variável (Ex.: n++): só adicionará +1 depois de terminar a operação. Exemplo:

```
int n = 4;
System.out.println ("O valor da variável é "+ n++); //o 1º símbolo (+) é a concatenação da string, o 2º (++) é um pós-incremento, então o programa vai imprimir que o valor da variável é 4, e só DEPOIS vai adicionar +1 à variável.
```

➤ Operadores de Atribuição:

+=	Somar e atribuir	a += b	a = a + b
-=	Subtrair e atribuir	a -= b	a = a - b
*=	Multiplicar e atribuir	a *= b	a = a * b
/=	Dividir e atribuir	a /= b	a = a / b
%=	Resto e atribuir	a %= b	a = a % b

O operador (+) também é usado para concatenar strings.

**Concatenação** é a operação de unir o conteúdo de duas strings. Por exemplo, considerando as strings "casa" + "mento" = "casamento".

O (%) calcula o resto da divisão. Observe no código a seguir:

v1 = 7, v2 = v1 % 2, v3 = 2; //7 dividido 2 sobra resto 1

v3 += v2; //essa atribuição significa v3 = v3 + v2

System.out.print(v1 + " " + v2 + " " + v3); //vai imprimir: 7 1 3

➤ Operadores Lógicos:

<b>&amp;&amp;</b>	.E.	true && false	false
<b>  </b>	.OU.	false    true	true
<b>^</b>	.XOU.	true ^ true	false
<b>!</b>	.NAO.	! false	true

- ✓ “**&&**” = And
- ✓ “**||**” = Or (são duas barras em pé)
- ✓ “**^**” = Xor (V+V=F // F+F=V)
- ✓ “**!**” = Not

//EXERCICIO COM PEGADINHA:

```
F           F           repare aqui
boolean val1 = (4>=5), val2 = (4<4), val3 = (val1==val2); // val1 e val2 tem a
mesma lógica booleana (é VERDADEIRO que F é igual a F)
          F ^ V = V   (^ é XOR)
boolean val4 = val1 ^ val3;
          V e V = V (AND)
boolean val5 = !val2 && val4;
System.out.println(val4 +" " + val5);    //vai imprimir TRUE TRUE
```

➤ Operadores Ternários:

São os **?** e **:** e tem esse nome porque são 3 operandos, eles equivalem ao **IF** e o **ELSE**, respectivamente:

o 1º operador / comando é uma expressão. Como em:  $(n1>n2)$

o 2º define o valor caso a expressão seja verdadeira

o 3º define o valor caso a expressão seja falsa. Exemplo:

$r = (n1>n2) ? 0 : 1;$

//Se  $n1 > n2$  for VERDADEIRO então r recebe 0, senão r recebe 1

## Aula 09

Estruturas condicionais **simples**: contém apenas o **IF**

Estruturas condicionais **compostas**: contém também o **ELSE**

É necessário sempre fechar o bloco para cada operando If/Else

As linhas com **condicional IF** só serão executadas se o **valor booleano** das sentenças (entre parênteses, como exemplo: a < b) for **VERDADEIRO**

```
int nasc = teclado.nextInt();
int i = 2015 - nasc;
if (i>=18) {
    System.out.print("Maior");
} else {
    System.out.print("Menor");
}
```

//não esquecer de abrir e fechar os blocos com as chaves

//EXERCICIO COM PEGADINHA:

A resposta é (B)

Dada a estrutura representada abaixo:

```
int a = 5, b = 2;
String c;
if (a > b) {
    c = "Primeiro é Maior";
} else {
    c = "Segundo é Maior";
}
```

Que linha substituiria a condição apresentada?

- a) c = (a < b)?"Segundo é Maior":"Primeiro é Maior";
- b) c = (a <= b)?"Segundo é Maior":"Primeiro é Maior";
- c) c = (a > b)?"Segundo é Maior":"Primeiro é Maior";
- d) c = (a >= b)?"Primeiro é Maior":"Segundo é Maior";

## Aula 10

Estruturas condicionais composta **Encadeada**: quando uma condicional está dentro da outra: **IF { } ELSE IF { }** “Se, Senão Se”  
São muito úteis para trabalhar com intervalos (exemplo: **2 a 5** && **9 a 14**)

//EXERCÍCIO COM PEGADINHA

Resposta: A

Observe o código Java abaixo:

```
public class Teste {  
    public static void main(String[] args) {  
        String nome = "João";  
        imprimeNome("Empty");  
    }  
    public static void imprimeNome(String nome) {  
        if(!nome.isEmpty()) {  
            System.out.println("Tudo bem " + nome + "?");  
        } else {  
            System.out.println("O nome é " + nome + "?");  
        }  
    }  
}
```

Qual será a saída do programa acima?

- a) Tudo bem Empty?
- b) Tudo bem João?
- c) O nome é Empty?
- d) O nome é João?
- e) Tudo bem Empty? O nome é João?

O método **.isEmpty**, dentro de uma String, vai **verificar se a String está vazia**

Existe diferença entre o método **.equals()** e o operador relacional **==**

O método **.equals(nome1.equals(nome3))**; **verifica se o conteúdo de um objeto é igual ao conteúdo do outro, mesmo que uma variável seja objeto instanciado e a outra não seja.**

O método **.charAt(0)** vai **verificar o caractere na posição 1**

//EXERCÍCIO COM PEGADINHA

Resposta: B

35. O que será impresso quando o seguinte programa escrito na linguagem JAVA for compilado e executado?

```
class Teste {  
    public static void main(String args[]) {  
        char ch;  
        String test2 = "abcde";  
        String test = new String("abcde");  
        if(test.equals(test2)) {  
            ch = (test == test2)? test.charAt(0) : test.charAt(1);  
        } else {  
            ch = (test == test2)? test.charAt(2) : test.charAt(3);  
        }  
        System.out.println(ch);  
    }  
}
```

a) a  
b) b  
c) c  
d) d  
e) e

Estruturas condicionais de **Múltipla Escolha**: quando há muitos casos (**Switch → Case → Break → Default**) “Escolha, caso”

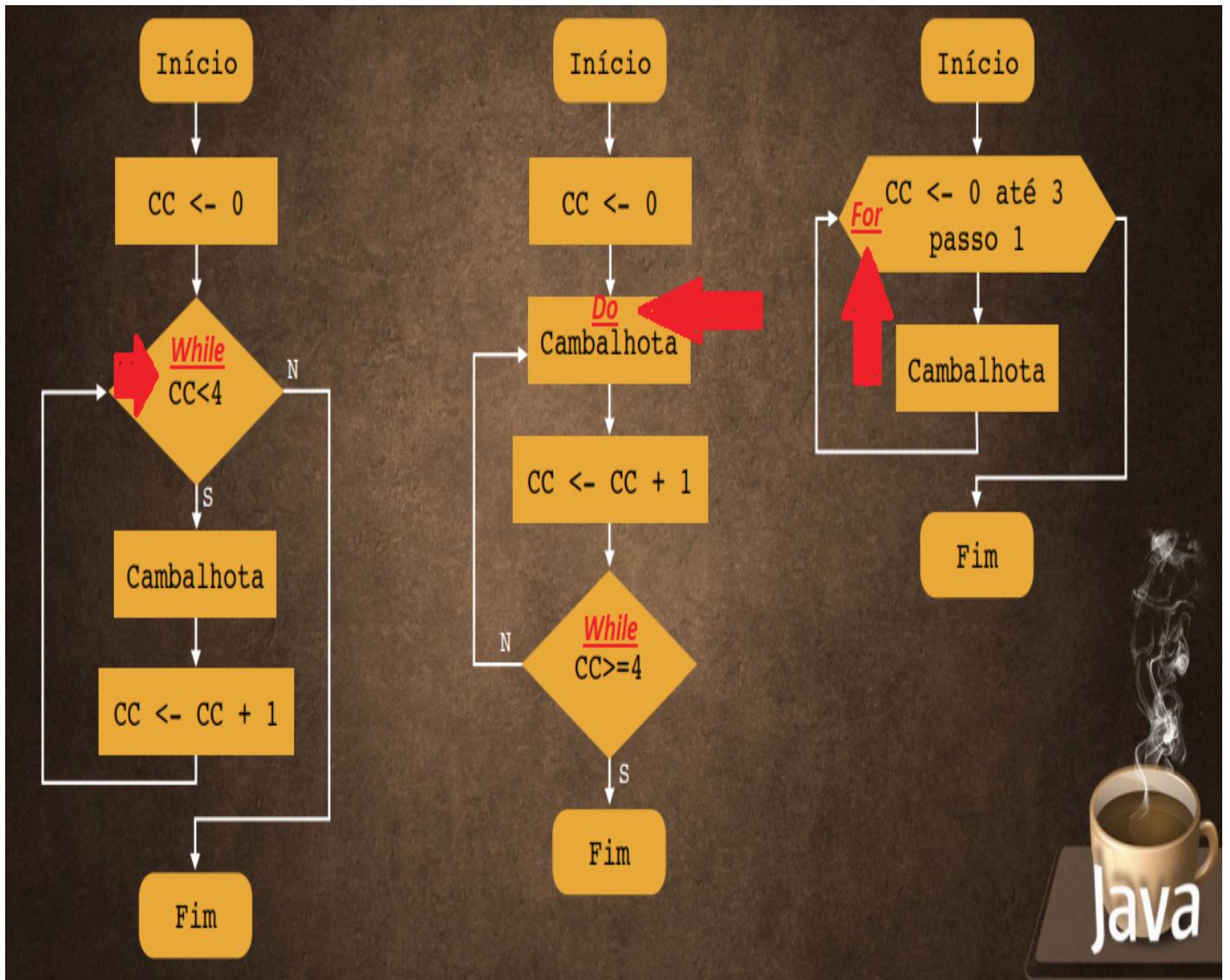
Não funciona por intervalos (Exemplo: de 2 a 9 é necessário declarar casos um por um)

- O **Default** é colocado no final da estrutura e responde quando nenhum dos casos foram atendidos.
- O **Break** no fim de Default é facultativo.

A estrutura **Switch** não serve para números Reais, apenas para inteiros

```
switch (perna) {  
    case 1:  
        tipo = "saci";  
        break;  
        ...  
        ...  
        ...  
    case 8:  
        tipo = "aranha gigante!";  
        break;  
    default:  
        tipo = "centopeia";  
        break;
```

Apresentando os 3 diferentes tipos de Iteração:



## Aula 11

A estrutura de repetição **while** (enquanto): Faz o teste lógico no início e repete quantas vezes forem necessário. Exemplo:

```
int cc = 0;
while (cc<4) {
sout....
cc++; //é importante o incremento senão ficará num loop eterno
}
```

//EXERCÍCIO COM PEGADINHA

Resposta: B e E estão corretas

Analise as seguintes variáveis em JAVA a seguir.

```
char c = 'c';
int i = 10;
double d = 10;
long l = 1;
String s = "Hello";
```

De acordo com as variáveis acima, qual das instruções abaixo compila sem erro?

- (A) `c = c + i;`
- (B) `s += i;`
- (C) `i += s;`
- (D) `c += s;`
- (E) `i += l;`

Algumas características dos tipos primitivos para entender a questão acima:

- A) char não pode receber inteiro
- B) string pode receber inteiro (quando concatenado)
- C) inteiro não pode receber string
- D) char não pode receber string
- E) int pode receber long (dependendo do valor do long)

**Modificar o fluxo de um laço** é mudar a ordem natural de uma repetição:

- O comando **continue** ignora o resto da repetição, retornando para a próxima iteração do loop (topo)
- Já o comando **break** faz o contrário, interrompe a repetição, indo para fora do laço.
- O comando **return** vai retornar um valor dentro de uma função.

```
public class Repeticao {
    public static void main(String[] args) {
        int cc = 0;
        while (cc<=10) {
            cc++;
            if (cc==3 || cc==4) {
                continue; // interrompe o fluxo e retorna para o topo do laço/loop
            }
            if (cc==8) {
                break; // já o 'break' faz o contrário, jogando para fora do loop/laço
            }
            System.out.println("cambalhota " + cc); }}
```

Aula 12

A estrutura de repetição **do** (equivale ao Repita): Faz o teste lógico no fim. É exatamente o oposto do While. É quase um while up-side-down. Exemplo:

```
int cc = 0;  
do {  
    System.out.println("cambalhota");  
    cc++;  
} while (cc >= 4); // note que o bloco é fechado antes da condição de repetição
```

#### //EXERCÍCIO COM PEGADINHA

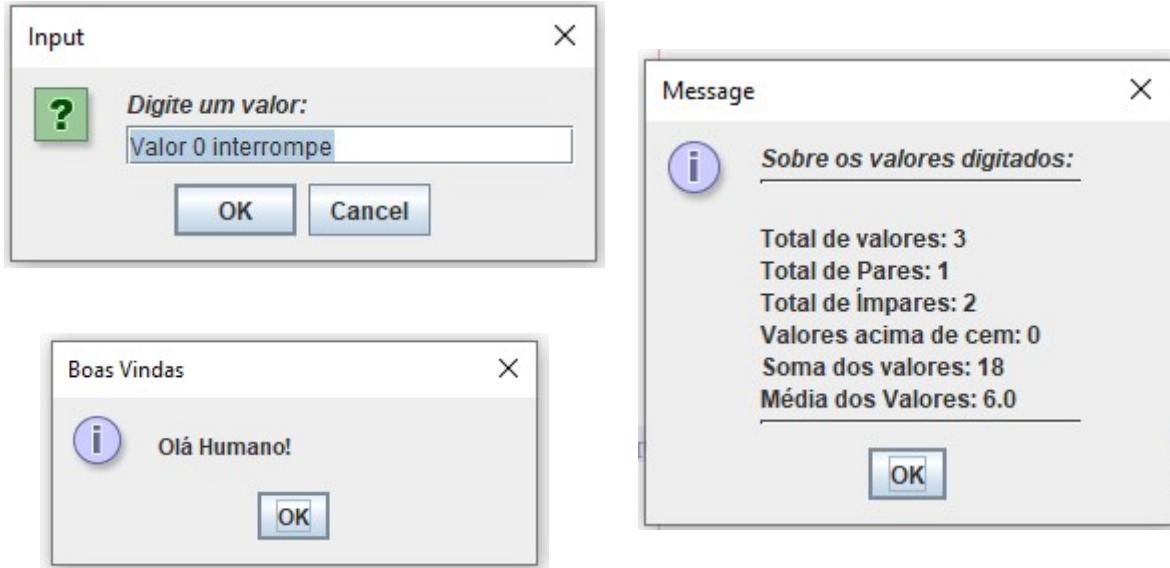
Qual será o resultado impresso pelo trecho de código escrito em Linguagem Java?

```
int c = 1;  
do {  
    if (c % 5 != 0) System.out.print(c);  
    else break;  
    c+=1;  
} while (c <= 10);
```

- a) 1 2 3 4 5 6 7 8 9 10
- b) 1 2 3 4 6 7 8 9
- c) 1 2 3 4
- d) Ocorrerá um erro de sintaxe

//Nesse código acima, imprimiu apenas os números: 1234 e encerrou.  
Mais um exemplo do comando break encerrando uma iteração

Dentro da biblioteca Swing temos **JOptionPanes**: são painéis/telas/minijanelas já prontas, como: mensagens de diálogo, informações, avisos, erros, telas para solicitar dados (assim como a função Scanner teclado()), etc. Exemplo:

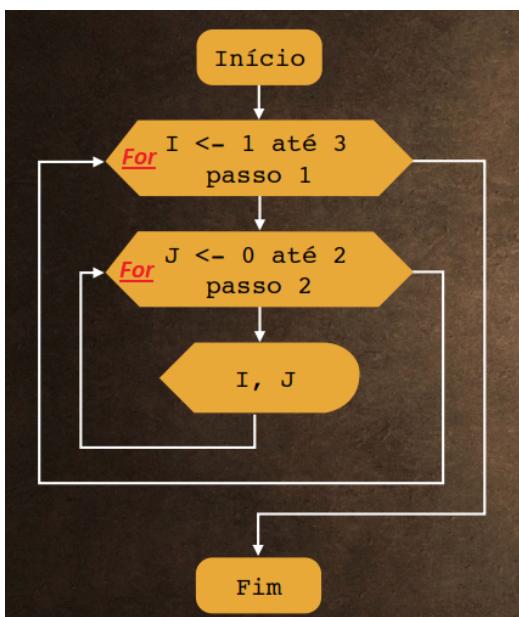


```
javax.swing.JOptionPane.showMessageDialog(null, "Olá Humano!", "Boas Vindas", javax.swing.JOptionPane.INFORMATION_MESSAGE);
```

## Aula 13

A estrutura de repetição **for** (É um Enquanto simplificado). Faz o teste lógico no meio da declaração. Ela é uma repetição com variável de controle, recebe a atribuição automaticamente até que seu valor estoure o limite (não precisa de incremento `++`), e se o passo estiver omitido então considera `+1`. Exemplo:

```
for (int cc = 0; cc <= 3; cc++) {  
    System.out.println(cambalhota);  
} // O 1º (;) declara o início do loop, o 2º (;) declara o “até”, e o 3º é o passo (de quanto em quanto)
```



**Laços Aninhados:** são um dentro do outro como se fosse um ninho.  
Em estruturas de repetição, quando temos um laço dentro outro, o retorno vai primeiro para o laço de dentro(J) para depois para o de fora(I).

**Iteração** significa repetir. \*Diferente de Interação

# Aula 14

As **variáveis compostas** são **Vetores** ou matrizes.

```
int vet[ ] = new int [4]    cria um vetor de 4 posições  
int [ ]vet = new int [4]    também pode ser feito assim
```

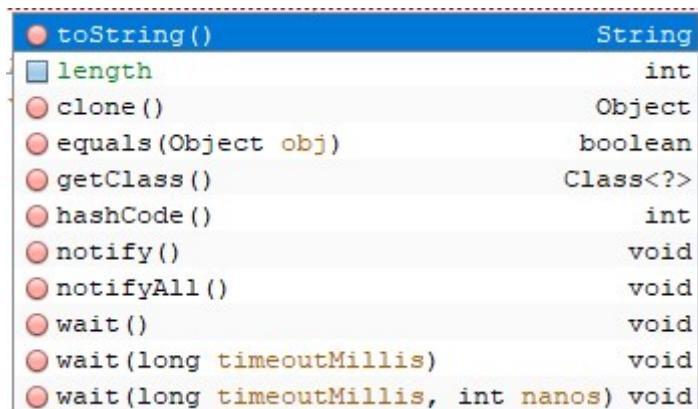
No Java, assim como no PHP, a primeira posição de um vetor é sempre 0.

vet [0] = 3 isso significa que 3 está no índice 0 do vetor n

Outra maneira simples de declarar um vetor, já inserindo valores:

```
int vet [ ] = {3,5,8,2}; //não esquecer do ponto-e-vírgula no fim
```

O vetor é um objeto, então ele tem métodos, propriedades e atributos.



A figura acima mostra várias funções de uma classe: **length** é um atributo e os demais são **métodos**.

**Métodos** são as ações que um objeto pode realizar, dividindo-se em funções ou procedimentos. Os **objetos** são características definidas pelas **classes**. E os **Atributos** são dados, parâmetros.

Existe uma **Iteração mais simples** e exclusiva para vetores, é a **for each** (significa para cada). Segue exemplo:

```
int vet [ ] = {3,5,1,8,4};  
for (int valore: vet) {  
    System.out.println(valore);  
}
```

//A variável `valore` (que deve ser do mesmo tipo do vetor) vai ser alimentada com os elementos de **vet**, começando do primeiro elemento(3) e indo até o último(8). E atenção para o **(:)**. “Para cada elemento de **vet**, coloque dentro de **valore**”

O Java é orientado a objeto e ele tem muitas classes prontas:  
A classe **Arrays** é a classe especializada em vetores. Segue alguns métodos da classe **Arrays**:

**Arrays.sort()**, ele ordena o vetor com apenas 1 linha de código

Existem vários **tipos de buscas**: sequencial, binária, de acesso aleatório...  
Uma busca binária é consideravelmente melhor do que busca sequencial.  
E para fazer uma busca é bem simples também:

```
int posicao = Arrays.binarySearch(vet,2); //esse método identifica a posição  
do elemento dentro do vetor, sendo o 1º parâmetro para o nome do vetor e o  
2º parâmetro para identificar qual o elemento buscar (key)
```

Lembrando que **BinarySearch** só pode ser usado em listas ordenadas!  
E se retornar um valor negativo, é porque não foi encontrada a chave(key)

**Arrays.fill()**, método para preenchimento automático do vetor:

```
int vet[ ] = new int [20];  
Arrays.fill(vet, 0);
```

//O 1º parâmetro determina o vetor escolhido, o 2º atribui o valor escolhido

**length()**, método que retorna o comprimento de uma string.

O exemplo abaixo mostra um Loop For Each com esse método:

```
for (int c = string.length()-1; c >= 0; c--) {
```

## Aula 15

### ➤ Classes:

Para criar uma classe basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves {}

### ➤ Visibilidade:

- A palavra **private** trava a visibilidade de um método, bloqueando seu acesso. A única classe que tem acesso ao atributo é a própria classe que o define.
- Já a palavra **public** serve para tornar o método visível/público a todos, liberando o acesso para qualquer um utilizar.

- O modificador **protected** torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

➤ **Rotinas:**

No Java, não há uma palavra específica para função ou procedimento, tudo é chamado de método, sendo diferenciado através do tipo de retorno. Então, para identificarmos um método, observamos seu início e verificamos se há a palavra void (que mostra ser um procedimento) ou um tipo primitivo (int, float, char, etc) o que, por sua vez, mostra ser uma função com retorno.

- ◆ **Procedimentos** são rotinas que não retornam valor. Para isso, usa-se a palavra void antes do método para declarar que o retorno é vazio. Segue exemplo de um procedimento:

```
void soma (int a, int b) {      //declarando um procedimento(void)
int s = a + b;           //declarando uma var (já aproveitando pra somar)
System.out.print("A soma é " + s); //string + concatenação de uma variável
}    //não esquecer de fechar o bloco
soma (5,2); // chama o procedimento, sendo 5 e 2 os atributos que vão assumir a e b.
```

- ◆ **Funções** são rotinas que retornam valor. Para isso, coloca-se um tipo primitivo antes do método para declarar que existe um retorno. Lembrando que funções podem ser parâmetros para outras funções. Segue exemplo de uma função:

```
int soma (int a, int b) { //declarando uma função (int)
int s = a + b;           //declarando uma var (já aproveitando pra somar)
return s;                //quando retornar s, retornará um valor inteiro
}    //não esquecer de fechar o bloco
int soma = soma(3,7); //uma das maneiras de chamar uma função é atribuindo o valor de retorno(s) a uma variável
```

## ➤ Explicando a função principal:

O método main é executado primeiro, até mesmo antes de qualquer procedimento no topo do programa.

**public static void main(String[ ] args) {** //main é o nome de um método que não retorna valor (por ser void), que recebe um vetor (de nome args) como parâmetro e, além disso, é também um método estático e público.

A palavra **static** indica que o método funciona apenas dentro de sua classe. Chamar um procedimento, dentro de um método estático, só é possível se esse procedimento for, também, estático. Com isso, não é preciso transformá-lo em objeto para poder utilizá-lo.

Reuso é uma das características muito importantes da POO.