

SENAI

<LAB365>

JAVA INICIAL - PARTE 1

<LAB365>

AGENDA

- Primeiros Passos no IntelliJ Community
- Sintaxe, Compilação e Execução
- Variáveis e Tipos de Dados
- Operadores
- Hands On

PRIMEIROS PASSOS NO INTELLIJ COMMUNITY



🕒 Mão na massa...



SINTAXE, COMPILAÇÃO E EXECUÇÃO

- Na linguagem Java, todo o código estará presente dentro de classes, que são arquivos com extensão *.java* e que contém tudo o que escrevemos;
- As classes são como moldes para criarmos representações de algo do mundo real;
- Quando criamos uma classe, sempre devemos nomeá-la com o nome exato do arquivo;

```
package base;

public class MyFirstClass {

    public static void main(String[] args) {
    }
}
```

SINTAXE, COMPILAÇÃO E EXECUÇÃO

- A linguagem possui um método chamado `main` que é o ponto de entrada da aplicação, ou seja, o método que o Java procura para executar. Ele possui uma nomenclatura específica e que deve ser seguida.
- A IDE (IntelliJ) possui atalhos para a construção do método `main`. Basta digitar `main` ou `psvm`, que a IDE oferecerá a opção para autocompletar o comando para você.
- Você pode escrever seu código dentro deste bloco, e tudo será executado.

```
public static void main(String[] args) {  
}
```

SINTAXE, COMPILAÇÃO E EXECUÇÃO

- **public** é o modificador de acesso do método. Ele informa que este método é acessível de qualquer outra classe dentro do aplicativo.
- **static** diz que este método é estático, ou seja, ele pertence à classe e pode ser chamado por meio dela, e não de uma instância.
- **void** é o tipo de retorno do método, ou seja, o que é retornado como resultado dele, que neste caso, por ser void, quer dizer que não terá retorno.
- **main** é o nome do método.
- **String[] args** é um parâmetro, que é uma matriz de Strings, que são textos.

```
public static void main(String[] args) {  
}
```

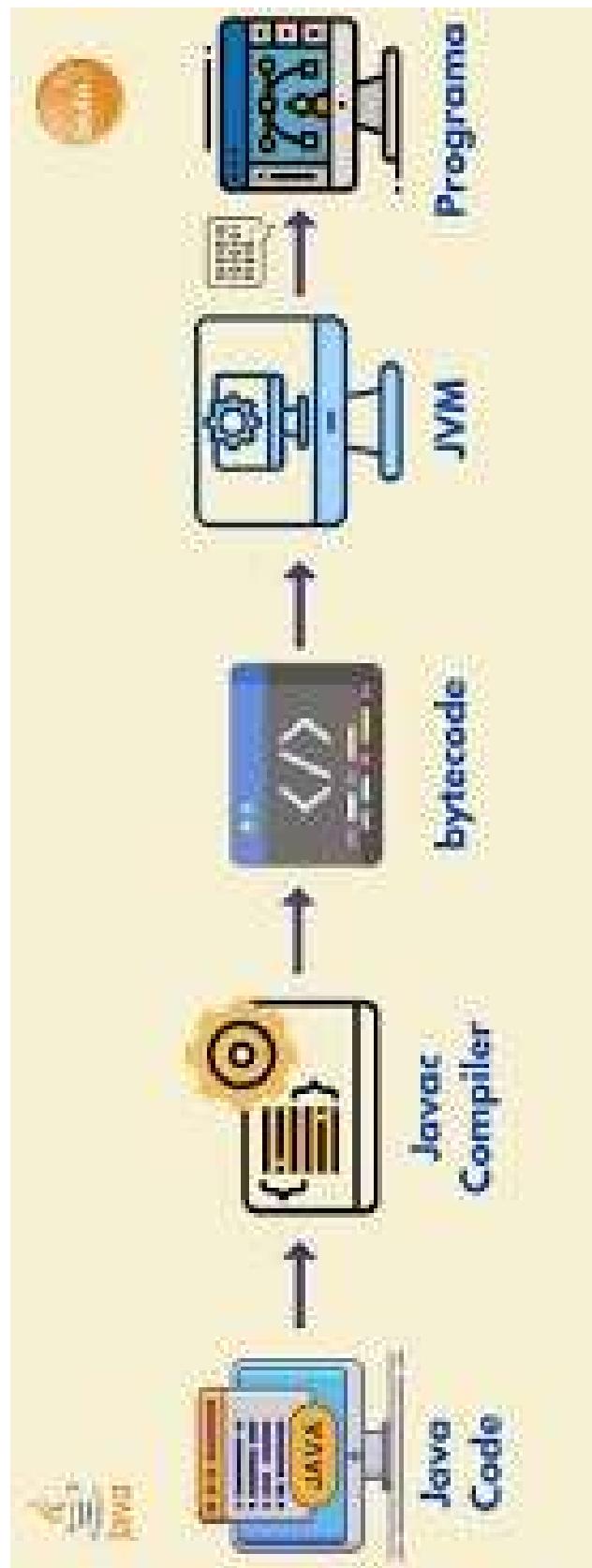
SINTAXE, COMPILAÇÃO E EXECUÇÃO

- **Compilação:** No Java o processo de compilação é basicamente a transformação do código escrito pelo desenvolvedor em um arquivo binário, denominado bytecode. Tendo em vista que a JVM não comprehende o texto em si, é necessário realizar essa conversão.
- **Execução:** A leitura do arquivo transformado na etapa de compilação, realizando assim o *start* do código desenvolvido.

SINTAXE, COMPILAÇÃO E EXECUÇÃO

<LAB365>

SENAI



VARIÁVEIS E TIPOS DE DADOS

- As variáveis são uma forma de armazenar dados em Java. Sendo a linguagem Java fortemente tipada, para definir uma variável é necessário declarar explicitamente um tipo e um nome para essa variável, ficando como opcional a inicialização.
- O nome de uma variável pode ser qualquer um, desde que não seja uma palavra reservada da linguagem. Porém, devemos sempre seguir as boas práticas para nomenclatura de variáveis.

VARIÁVEIS E TIPOS DE DADOS

Constantes, diferentes de variáveis, são imutáveis e uma vez que tenha sido declarada, não pode haver troca de valores.

- Para criar uma constante, usa-se a palavra-chave final.
- Por convenção, nomes de constantes têm sua declaração com letras maiúsculas.

```
final int MY_CONSTANT = 100;
```

VARIÁVEIS E TIPOS DE DADOS

abstract	class	extends	implements	null	strictfp
true	assert	const	false	import	package
super	try	boolean	continue	final	instanceof
Private	switch	void	break	default	finally
int	protected	synchronized	volatile	byte	do
float	interface	public	this	while	case
double	for	long	return	throw	catch
else	goto	native	short	throws	char
enum	if	new	static	transient	

VARIÁVEIS E TIPOS DE DADOS

Pascal Case		PascalCase
Snake Case		snake_case
Camel Case		camelCase
Screaming Snake Case		NOME_DA_VARIAVEL
Kebab Case		nome-da-variavel

VARIÁVEIS E TIPOS DE DADOS

in that case...



Fonte da imagem: <http://visualdicas.blogspot.com/2021/05/quais-as-formas-mais-populares-para.html>

VARIÁVEIS E TIPOS DE DADOS

Pascal Case		definição de classes
Snake Case		-
Camel Case		definição de métodos, atributos, parâmetros, ...
Screaming Snake Case		definição de constantes e valores de enum
Kebab Case		-

VARIÁVEIS E TIPOS DE DADOS

- **Atenção:** Java é uma linguagem Case Sensitive.

Para nome de variáveis:

- Pode começar com letra ou " _ ",
- Pode utilizar números;
- Utilize nomes descriptivos;
- NÃO pode começar com números;
- NÃO recomendável utilizar acento;
- NÃO pode conter espaços;
- NÃO usar palavras reservadas (int, return, void, null, ...)

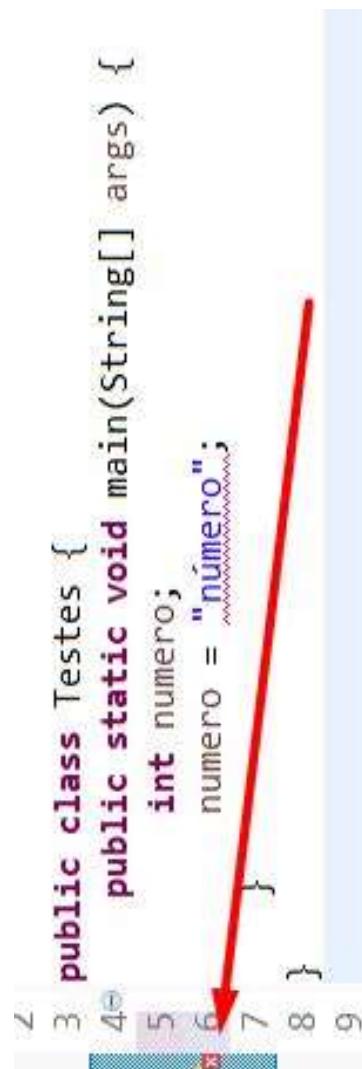
Obs: O cuidado com as convenções torna o código de fácil interpretação.

VARIÁVEIS E TIPOS DE DADOS

- Em uma linguagem fortemente tipada toda a variável possui um tipo, toda expressão possui um tipo e todo tipo é bem definido. Qualquer atribuição com tipos conflitantes gera um erro em tempo de compilação.
- Para entender as linguagens de tipagem fraca é simples, essas linguagens não possuem tipos bem definidos e as atribuições feitas de forma indevida não geram erro.

```
1  var numero;
2
3  numero = 1;
4  numero = "Número";
```

```
1
2
3 public class Testes {
4     public static void main(String[] args) {
5         int numero;
6         numero = "número";
7     }
8 }
9
```



VARIÁVEIS E TIPOS DE DADOS



Sempre lembre-se que são oito tipos primitivos definidos em java e quatro grupos onde esses oito tipos estão distribuídos!!!

Inteiros	byte, short, int, long
Números com ponto flutuante	float, double
Caracteres	char
Booleanos (lógicos)	boolean

VARIÁVEIS E TIPOS DE DADOS

Dos tipos primitivos definidos em Java, quatro são do tipo inteiro. Basicamente o que deve ser levado em consideração para a escolha do tipo inteiro a ser utilizado no código é a sua capacidade de armazenamento de dados.

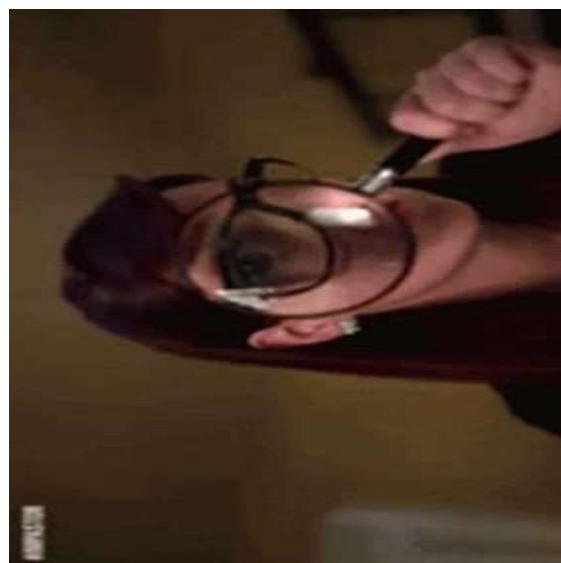
Nome	Largura	Valores máximo e mínimo
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
int	32 bits	-2.147.483.648 a 2.147.483.647
short	16 bits	-32.768 a 32.767
byte	8 bits	-128 a 127



VARIÁVEIS E TIPOS DE DADOS

Dois tipos primitivos em Java são considerados de ponto flutuante. São eles: float e double. O que deve ser levado em consideração na hora de escolher qual tipo utilizar é o grau de precisão necessário para representar o que se deseja armazenar. O tipo float possui uma menor precisão em relação ao tipo double.

Nome	Largura	Faixa aproximada
double	64 bits	4.9e-324 a 1.8e+308
float	32 bits	1.4e-045 a 3.4e+038



VARIÁVEIS E TIPOS DE DADOS

- O tipo primitivo em Java utilizado para armazenar caracteres é o char, que em Java tem um comprimento de 16 bits. para atribuir um valor a um atributo do tipo char utiliza-se aspas simples.
- EX: `char letra = 'a';`

VARIÁVEIS E TIPOS DE DADOS

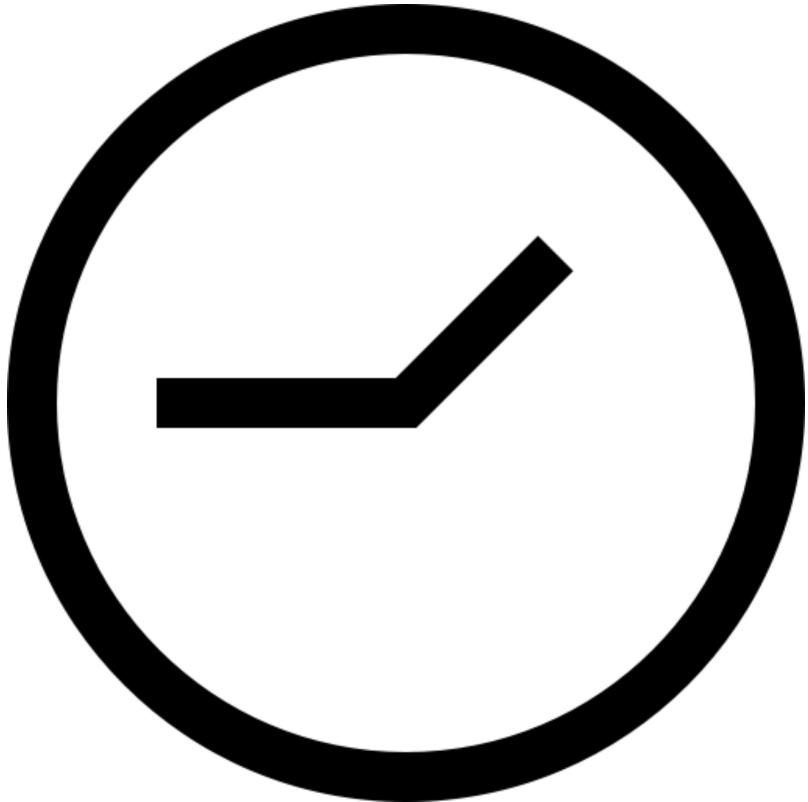
- O tipo boolean é utilizado para valores lógicos, existem apenas duas possibilidades de valores para esse tipo: false ou true. Esse tipo é retornado em todas as operações de comparação.
 - EX: $10 > 9$.  true.
 - São obrigatórios para operações condicionais e que gerenciam instruções de controle (if e for).

VARIÁVEIS E TIPOS DE DADOS

```
public static void main(String[] args) {  
    boolean b;  
  
    b = false;  
    System.out.println("b é " + b);  
    b = true;  
    System.out.println("b é " + b);  
  
    // Um valor boolean pode controlar a instrução if  
    if (b) {  
        System.out.println("Isso é executado.");  
    }  
    b = false;  
    if (b) {  
        System.out.println("Isso não é executado.");  
    }  
  
    // O resultado de um operador relacional é um valor boolean  
    System.out.println("10 > 9 é " + (10 > 9));  
}
```

VARIÁVEIS E TIPOS DE DADOS

- **Tipos não primitivos**
 - Caso não inicializados, o valor é nulo.
 - São tipos de referência (String e Objects)



INTERVALO!

Finalizamos o nosso
primeiro período de hoje.
Que tal descansar um
pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40

OPERADORES

Dentre os operadores disponíveis na linguagem Java, estes são os mais populares:

- Operadores aritméticos
- Operadores de atribuição
- Operadores unários
- Operadores de igualdade e relacionais
- Operadores lógicos
- Operador ternário

OPERADORES

- Adição: **+** (Somar valores)
- Subtração: **-** (Subtrair valores)
- Multiplicação: ***** (Multiplicar valores)
- Divisão: **/** (Dividir valores)
- Módulo: **%** (Obter o resto da divisão entre dois números)

OPERADORES

- Atribuição simples: = (int a = 10;)
- Atribuição composta: +=, -=, *=, /=

A atribuição composta é uma forma simplificada de atribuir valores em variáveis. Ela funciona da seguinte forma:

```
int a = 10;  
a = a + 2; // 12  
a += 2; // o mesmo que a = a + 2, mas simplifica o código.
```

OPERADORES

- Operador de igualdade: ==
- Operador de diferença: !=
- Operador maior que: >
- Operador maior ou igual a: >=
- Operador menor que: <
- Operador menor ou igual a: <=

OPERADORES

- Operador AND: && (A condição é atendida se todas as sentenças forem verdadeiras)
- Operador OR: || (A condição é atendida se pelo menos uma sentença for verdadeira)
- Operador NOT: !(Retorna a negação da afirmação. Se é verdadeiro, torna-se falso)

Os operadores lógicos são muito utilizados em condicionais. O de negação pode ser usado quando queremos verificar se algo é falso. Por exemplo, queremos saber se uma pessoa é menor de idade:

```
int idade = 20;
```

```
boolean maiorDeIdade = idade >= 18;  
  
if (!maiorDeIdade) { // Lê-se "se não for maior de idade", ou "se  
maiorDeIdade for igual a false"  
}
```

OPERADORES

O operador ternário é uma forma simplificada de fazer uma condicional simples (*if*) em casos onde temos sentenças simples. Sua estrutura é a seguinte:

```
int valor = 10;  
String resultado = (valor > 10) ? "Maior que 10" : "Menor que dez";
```

↙ Mão na massa...



HANDS ON

SENAI

<LAB365>

JAVA INICIAL - PARTE 2

<LAB365>

AGENDA

- Estrutura de Decisão
- Estrutura de Escolha

ESTRUTURA DE DECISÃO

- As estruturas de decisão em Java permitem controlar o fluxo do seu programa com base em determinadas condições. Essas estruturas possibilitam a execução de blocos de código diferentes, dependendo se uma condição especificada avaliada como verdadeira ou falsa.
- Em Java, as estruturas de decisão são principalmente implementadas usando as instruções if, else if e else.

ESTRUTURA DE DECISÃO

- A instrução if é a estrutura básica de tomada de decisões. Ela avalia uma condição e, se verdadeira, executa um bloco de código.

```
if (condição) {  
    // Código a ser executado se a condição for verdadeira  
}
```

ESTRUTURA DE DECISÃO

- A instrução if-else permite executar um bloco de código se a condição for verdadeira e outro bloco se a condição for falsa.

```
if (condição) {  
    // Código a ser executado se a condição for verdadeira  
} else {  
    // Código a ser executado se a condição for falsa  
}
```

ESTRUTURA DE DECISÃO

- A instrução else if é usada em conjunto com a instrução if para avaliar várias condições em sequência.

```
if (condição1)
    // Código a ser executado se a condição1 for verdadeira
} else if (condição2) {
    // Código a ser executado se a condição2 for verdadeira
} else {
    // Código a ser executado se todas as condições anteriores forem falsas
}
```

ESTRUTURA DE DECISÃO

Exemplo Prático:

```
int numero = 10;

if (numero > 0) {
    System.out.println("O número é positivo.");
} else if (numero < 0) {
    System.out.println("O número é negativo.");
} else {
    System.out.println("O número é zero.");
}
```

Essas estruturas de decisão permitem que o seu código tome diferentes caminhos com base nas condições definidas, tornando o seu programa mais flexível e adaptável

ESTRUTURA DE DECISÃO

EXERCÍCIO 01

Verificação de Números Pares e Ímpares:

- Crie um programa em Java que solicita ao usuário para inserir um número inteiro.
- Use uma estrutura de decisão para determinar se o número é par ou ímpar.
- Exiba uma mensagem indicando se o número é par ou ímpar.

Classificação de Idades:

- Escreva um programa que pede ao usuário para inserir a idade.
- Utilize estruturas de decisão para classificar a idade em categorias: "Criança" (0-12 anos), "Adolescente" (13-19 anos), "Adulto" (20 anos ou mais).
- Imprima a categoria correspondente.

EXERCÍCIO 02

ESTRUTURA DE DECISÃO

EXERCÍCIO 03

Calculadora de Bônus Salarial:

- Desenvolva um programa que solicita ao usuário para inserir o salário mensal e o tempo de serviço em anos.
- Use estruturas de decisão para calcular um bônus salarial com base nas seguintes regras:
 - Se o tempo de serviço for superior a 5 anos, o bônus é 10% do salário.
 - Caso contrário, o bônus é 5% do salário.
 - Exiba o salário original, o bônus calculado e o novo salário após o bônus.

ESTRUTURA DE DECISÃO

Dica:

- Use a classe Scanner para obter entrada do usuário nos programas.
- Utilize as estruturas de decisão (if, else if, else) conforme a lógica de cada exercício.
- Ao imprimir mensagens, torne-as informativas para o usuário entender o resultado.

Estes exercícios ajudarão os estudantes a praticar a implementação de estruturas de decisão em Java, fortalecendo suas habilidades de lógica de programação.

ESTRUTURA DE ESCOLHA

- **switch - case**
 - A estrutura switch case em Java é uma forma de controle de fluxo que permite selecionar um bloco de código para executar com base no valor de uma expressão.

```
switch (expressao) {  
    case valor1:  
        // Código a ser executado se expressao == valor1  
        break;  
    case valor2:  
        // Código a ser executado se expressao == valor2  
        break;  
    // Mais casos...  
    default:  
        // Código a ser executado se nenhum caso corresponder  
}
```

ESTRUTURA DE ESCOLHA

- **switch - case**
 - expressão é a expressão cujo valor será comparado com os casos.
 - case valor1: é case valor2: são os rótulos de caso. Se expressão for igual a valor1, o código dentro desse bloco será executado.
 - break; é usado para sair do switch depois a execução de um caso.
 - default: é opcional e é executado se nenhum dos casos corresponder.
- **Observação**
 - A expressão no switch deve ser do tipo char, byte, short, int, String ou um enum.
 - Os rótulos de caso devem ser constantes (literais ou final variables).

ESTRUTURA DE ESCOLHA

```
int diaDaSemana = 3;
switch (diaDaSemana) {
    case 1:
        System.out.println("Segunda-feira");
        break;
    case 2:
        System.out.println("Terça-feira");
        break;
    case 3:
        System.out.println("Quarta-feira");
        break;
    default:
        System.out.println("Dia desconhecido");
}
```

ESTRUTURA DE ESCOLHA

- Crie um programa em Java que solicite ao usuário o seu nível de condicionamento (iniciante, intermediário ou avançado) e, em seguida, gere um plano de treinamento semanal com base nesse nível.

Escolha o nível de condicionamento:

1. Iniciante
2. Intermediário
3. Avançado

Opção :

SENAI

<LAB365>

JAVA INICIAL - PARTE 3

<LAB365>

AGENDA

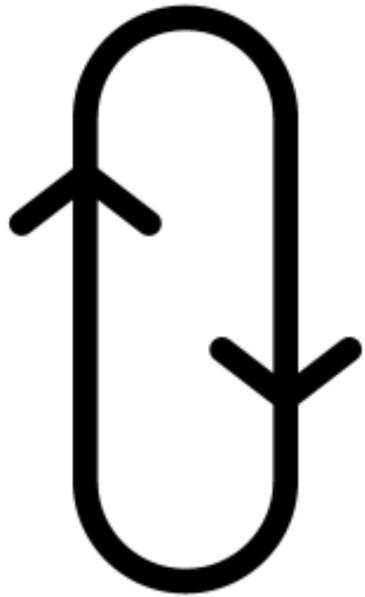
<LAB365>

SENAI

- Estrutura de Repetição
- Tira-Dúvidas

ESTRUTURA DE REPETIÇÃO

- Estrutura de repetição também são conhecidas como laços de repetição (loops)
- Executam instruções enquanto a condição determinada for verdadeira.
 - Essas condições determinadas são **expressões booleanas** avaliadas a cada laço.



ESTRUTURA DE REPETIÇÃO

- WHILE
 - Traduzido do inglês: "enquanto"
 - O laço "while", executa instruções enquanto a condição for verdadeira
 - Como usar um "while"?

```
int i = 0;  
while (i < array.length) {  
    // corpo  
    i++;  
  
while (<condicao>) {  
}  
while (true) {  
}
```

Mão na massa...



HANDS ON

ESTRUTURA DE REPETIÇÃO

- Escreva no console todos os números PARES de 1 até 100.
- Escreva no console todos os números ÍMPARES de 1 até 100.

ESTRUTURA DE REPETIÇÃO

- **DO-WHILE**

- Traduzido do inglês:
 - do: faça
 - while: enquanto
- O laço "do-while", é uma estrutura de repetição que tem uma sutil variação da estrutura "while".
 - No "while" a condição é testada antes de entrar no loop
 - No "do-while" a condição é testada após a primeira execução
 - Ou seja, sempre será executada uma vez antes de verificar se a condição determinada é verdadeira.

ESTRUTURA DE REPETIÇÃO

- DO-WHILE

```
do {  
    int i = 0;  
    do {  
        // corpo  
        i++;  
    } while (i < array.length);  
}  
} while (<condicao>);
```

↙ Mão na massa...



HANDS ON

ESTRUTURA DE REPETIÇÃO

- Crie um número aleatório, num intervalo de 0 à 10 e peça ao usuário para que adivinhe o número.
 - Sómente deverá sair do loop caso o usuário acerte o número.

ESTRUTURA DE REPETIÇÃO

O loop for é uma for versátil e poderosa de controlar as interações em Java. Ele é composto por três partes: primeira a inicialização de uma variável de controle, em seguida a condicional de controle que deve retornar um boolean e a variação da variável da controle.

```
public static void main (String [] args) {  
    for (int i=0; i < 3; i++) {  
    }  
}
```

Inicialização da variável de controle. É executada apenas uma vez que é no início da instrução for.

Condicional de controle que é verificada várias vezes ao longo da execução da instrução e deve sempre retornar um boolean

variação da variável de controle que sempre será modificada após o término do bloco de código contido no for. Ele pode ser incrementado i++ ou decrementado i--

ESTRUTURA DE REPETIÇÃO

```
public static void main(String[] args) {  
    for (int i=0; i < 3; i++) {  
    }  
}
```

Repare que é utilizado ; para separar as estruturas no loop for.



Mão na massa...



HANDS ON

ESTRUTURA DE REPETIÇÃO

- Escreva um programa utilizando a estrutura de repetição for que imprime no console números de 1 a 5.
- Escreva um programa que imprima uma tabuada de multiplicação utilizando a estrutura de repetição for. Ele deve perguntar ao usuário qual o número que ele deseja saber a tabuada.

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique [aqui](#) ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.



<LAB365>

<LAB365>

SENAI