

# Aula 3 - Parte 1

## Apresentação

# Dinâmica de análise de código

## Orientações:

- Escrevam os códigos dos exercícios feitos em aula e façam alterações que gerem erros (façam parar de rodar ou gerem warnings de propósito!).
- Podem ser criativos, troquem nomes, posições, tipos de variáveis, existem inúmeras formas de gerar erro!
- Quando gerarem o erro busquem o motivo dele e se façam as seguintes perguntas: Por que não posso fazer isso? Qual a solução desse problema?
- Quando se fizerem as perguntas, tentem responder, pesquisem possíveis respostas.

# Novidade, nem

## O que iremos aprender

- Variáveis Primitivas e Controle de Fluxo:
  - Declarando e usando variáveis;
  - Tipos primitivos e valores;
  - Casting;
  - O if e else (se, senão);
  - O while ( enquanto );
  - O for ( para... até... faça );
  - Escopo de variáveis;

# Type Casting

O que acontece com o seguinte trecho de código?

```
double d = 3.1415;  
int i = d;  
System.out.println("O valor de i é = " + i);
```

**Exception in thread "main"**  
**java.lang.Error: Unresolved compilation**  
**problem:**  
**Type mismatch: cannot convert from double to int**

# Type Casting

O que acontece com o seguinte trecho de código?

```
double d = 3.1415;  
int i = (int)d;  
System.out.println("O valor de i é " + i);
```

Saida : O valor de i é 3

# Type Casting

## Casting possíveis

- A indicação **Impl.** quer dizer que o cast é implícito e automático, ou seja, você não precisa indicar o cast explicitamente.
- O tipo **boolean** não pode ser convertido para outro tipo.

PARA:	byte	short	char	int	long	float	double
DE:	byte	short	char	int	long	float	double
byte	----	<i>Impl.</i>	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
short	(byte)	----	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
char	(byte)	(short)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
int	(byte)	(short)	(char)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
long	(byte)	(short)	(char)	(int)	----	<i>Impl.</i>	<i>Impl.</i>
float	(byte)	(short)	(char)	(int)	(long)	----	<i>Impl.</i>
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

# Exercícios

## Testando castings

- Declare duas variáveis do tipo **int** e realize sua soma.
  - Em seguida, realize o casting destes dois inteiros para **double** para realizar sua divisão.
- Declare dois caracteres : “A” e “Z”, depois realize sua soma e armazene em uma variável do tipo **int**.
  - Qual é o resultado apresentado?
  - Por que você acha que esse foi o resultado apresentado?



# Recordar é viver... e

## Desvios condicionais

- No nivelamento vimos que podemos desviar a execução de nosso código utilizando **se... senao**.

```
se (condicao) {  
    // Execute uma parte de código  
}  
senao {  
    // Execute outra parte de código  
}
```

# Recordar é viver... e

## Desvios condicionais

- Em Java para desviar a execução basicamente trocamos o **se** por **if** e o **senao** por **else**.

```
if (condicao) {  
    // Execute uma parte de código  
}  
else {  
    // Execute outra parte de código  
}
```

Só isso !?

# Recordar é viver... e

## Operadores lógicos

- A notação dos operadores lógicos que aprendemos anteriormente também muda:
- O operador **E** é representado por **&&**;
- O operador **OU** é representado por **||**;
- O operador **NAO** é representado por **!**;

```
if ( (A || B) && (C == D)    && !E ) {  
    // Execute uma parte de código  
}
```

# Recordar é viver... e

## Laços de repetição - **faca enquanto**

- Vimos a estrutura do **faca... enquanto** no nivelamento.

```
faca {  
    // Execute uma parte de código  
}  
enquanto (condicao)
```

# Recordar é viver... e

## Laços de repetição - **do while**

- Em Java basicamente trocamos o **faca** por **do** e o **enquanto** por **while**.

```
do {  
    // Execute uma parte de código  
}  
while (condicao)
```

# Recordar é viver... e

## Laços de repetição - enquanto

- Vimos a estrutura do **enquanto** no nivelamento.

```
enquanto (condicao) {  
    // Execute uma parte de código  
}
```

# Recordar é viver... e

## Laços de repetição - enquanto

- Em Java basicamente trocamos o enquanto por while.

```
while (condicao) {  
    // Execute uma parte de código  
}
```

# Recordar é viver... e

## Laços de repetição - **para**

- Vimos a estrutura do **para** no nivelamento.

```
para (inicializacao; condicao; incremento) {  
    // Execute uma parte de código  
}
```



# Recordar é viver... e

## Laços de repetição - **for**

- Em Java basicamente trocamos o **para** por **for**.

```
for (inicializacao; condicao; incremento)
{
    // Execute uma parte de código
}
```

Não é possível!  
Só isso mesmo !?






# Escopo de

- O que define um escopo?
  - Basicamente o uso de chaves `{ }`
- No Java, podemos declarar variáveis a qualquer momento. Porém, dependendo de onde você as declarou, ela vai valer de um determinado ponto a outro;
- Escopo da variável é o nome dado ao trecho de código em que aquela variável existe e onde é possível acessá-la;
- Quando abrimos um novo bloco com as chaves, as variáveis declaradas ou inicializadas ali dentro só valem até o fim daquele bloco.

# UTILIZANDO O ECLIPSE - ALGO MAIS...

## Pastas do Projeto

Dentro do diretório como o nome do projeto onde o Workspace foi criado temos a estrutura de pastas. abaixo:

 .settings	24/07/2020 10:20	Pasta de arquivos	
 bin	25/07/2020 11:09	Pasta de arquivos	
 src	24/07/2020 23:28	Pasta de arquivos	
 .classpath	24/07/2020 10:20	Arquivo CLASSPA...	1 KB
 .project	24/07/2020 10:20	Arquivo PROJECT	1 KB

- A pasta bin contém os arquivos .class
- A pasta src os arquivos .java
- O arquivo .classpath serve para informar onde serão armazenados os arquivos .class e .java
- O arquivo .project é utilizado pelo eclipse para configurações referente ao projeto.



Para visualizarmos a estrutura de pastas no Eclipse pressione CTRL+3 digite **navigator**

# BOAS PRÁTICAS E CONVENÇÕES

- **Pacotes:** eles devem ser escritos de forma semelhante a um endereço web, só que de trás para frente e ao final, indicamos um nome (ou um conjunto de nome), que classifica as classes agrupadas. (Ex.: “br.com.serratec.model”, “br.com.serratec.view”)
- **Classes e Interfaces:** nomes das classes e interfaces iniciam com uma letra maiúscula, sendo simples e descritivo. Caso seja nome composto utiliza-se o padrão *CamelCase*. (Ex.: “Usuario”, “ContaCorrente”)
- **Métodos:** os métodos seguem o mesmo padrão das classes, com a diferença que a primeira letra é minúscula. Como os métodos executam alguma ação, procure usar verbos para seu nome. (Ex.: “imprimirValor”, “executar”, “calcularMedia”)
- **Variáveis:** a convenção é a mesma adotada para métodos, com nomes curtos e significativos (ex.: “nome”, “nota”, “mediaAluno”). Evitar variáveis com apenas um caracter, a não ser que seja índice em repetições ou vetores (Ex.: “x”, “y”, “i”). Em constantes todas as letras deve estar em maiúsculas e separadas por “\_” (Ex.: “JUROS”, “DATA\_CORTE”).

# Revisando

## Já vimos um pouco mais sobre:

- O que é Java;
- Eclipse IDE;
- Nosso primeiro código em Java : “Olá Mundo!”;
- Variáveis e controle de fluxo;
- Declarando e usando variáveis;
- Tipos primitivos, valores e casting;
- O if e else (se, senão);
- O while e for ( enquanto, para... até);
- Escopo de variáveis;

# Pesquisa Complementar - 11/04/22

A Linguagem Java possui características importantes.

**Vamos fixar os conceitos e trazer para a turma...**

**Apresentação da pesquisa (formato simples, em torno de 5 minutos de fala por grupo).**

- Grupo 1 - Fortemente Tipada;
- Grupo 2 - Polimorfismo;
- Grupo 3 - Portabilidade e Herança;
- Grupo 4 - LGPD e Encapsulamento;
- Grupo 5 - Alta Performance;
- Grupo 6 - Interpretada.

**Lembrem-se: fala curta não significa pouca pesquisa! Pesquisem em várias fontes e conversem em grupo.**

# Pesquisa Complementar - 11/04/22

Hoje vimos alguns dos outros conceitos importantes!

**Vamos pesquisar mais sobre eles e trazer para a turma...**

**Apresentação da pesquisa (formato simples, em torno de 5 minutos de fala por grupo).**

- Grupo 1 - Pacotes e Classes; Estrutura do código: class;
- Grupo 2 - Objetos; Estrutura do código: public e syso;
- Grupo 3 - Interfaces; Estrutura do código: static;
- Grupo 4 - Métodos; Estrutura do código: void;
- Grupo 5 - Variáveis; Estrutura do código: main;
- Grupo 6 - Atributos; Estrutura do código: String[] args.

**Lembrem-se: fala curta não significa pouca pesquisa! Pesquisem em várias fontes e conversem em grupo.**