**Java JRE e JDK: Escreva o seu primeiro codigo com Eclipse**

01 – O que é Java

- características:

- orientado a objeto (O. O.)

- muitas bibliotecas

- semelhante ao C++

- roda e diversos SO

- O Java além do descrito acima também traz:

- portabilidade

- fácil acesso e desenvolvimento

- segurança

- onipresença

- James Gosling, considerado o pai do Java, apesar do mesmo ter sido criado por um grupo

- antigamente cada dispositivo tinha seu codigo fonte, o que necessitava de uma linguagem própria para funcionar, cada mudança no hardware exigia que o codigo fonte fosse reescrito

- foi então que surgiu a ideia de escrever um codigo que gerasse um executável (entre aspas), porque após a compilação ele estará em um formato que não sera compreendido pelo aparelho em si e sim por um intermediário, este intermediário simula um hardware simples que compreende o codigo que foi gerado, então surgiu a MAQUINA VIRTUAL (VIRTUAL MACHINE)

- a ideia inicial era criar um hardware que pudesse ser instalado em qualquer dispositivo, mas a ideia fracassou pois fabricar um chip para cada aparelho era algo custoso

- um dos problemas que a linguagem Java veio resolver era por exemplo um codigo que era escrito para Windows era compilado se tornando um .exe, porem só funcionaria nesse ambiente, e era dependente de outras bibliotecas e muitas vezes só era compatível com uma determinada versão do Windows

- com um codigo fonte único temos um intermediário que “traduz” ou “instrui” o SO sobre os comandos recebidos, esse intermediário é a JVM (Java Virtual Machine), que não é meramente um interpretador

- o codigo seria a linguagem Java

- quando compilado não geraria um .exe (pois este só pode ser entendido pelo Windows)

- o formato gerado é um Bytecode Java, extensão .class

- este sim reconhecido e processado pela JVM passando as instruções para o SO especifico

- o Java nasceu com um proposito, mas se fortaleceu no server-side, pois quando escrevemos uma aplicação, um site web, ou sistema grande não queremos ficar dependentes de diferentes Sistemas Operacionais

- as grandes empresas logo aderiram a plataforma Java pois não queriam ficar presas a determinada linguagem (Vendor lock-in)

- principais características da JVM

- multiplataforma

- gerenciamento de memoria

- segurança

- sandbox

- otimizações

- JIT compiler

- a plataforma (ou ecossistema) Java é interessante para as empresas pois não dependem do que esta abaixo na sua Stack

- diversas linguagens (Ruby, Python, Clojure, Scala, etc) possuem programas que geram bytecodes Java, e posteriormente a JVM trabalhara com esse codigo de acordo com o SO desejado

P: Qual é o grande beneficio da JVM?

R: Executar o codigo independente do SO.

P: Quais são algumas características do Java que são mantidas até hoje?

R: - Semelhante ao C++

- Muitas bibliotecas

- Multiplataforma (roda em qualquer SO desde que exista a JVM)

P: Quais sistemas podemos criar utilizando Java?

R: Sistemas Web (Web Sites, web app), sistemas server side, app para android, applets que rodam em navegadores.

P: Qual a diferença entre o executável Windows (.exe) e o do Java (bytecode)?

R: O executável do Java é portátil o do Windows não, os executáveis do Windows podem ser executado diretamente no SO, o do Java necessita de uma JVM.

- algumas afirmações sobre os bytecodes.

- é semelhante ao assembly

- é executado pela JVM

- é independente do SO

- é incorreto afirmar que o bytecode é o codigo fonte em si

- existe um conjunto de comandos que a JVM entende, esses comandos são chamados de opcodes (operation code), cada opcode possui o tamanho exato de 1 byte, ou seja temos um opcode de 1 byte ou bytecode

02 – Instalaçao e o primeiro programa

- utilizaremos o JDK (Java Development Kits)

- existe o JRE (Java Runtime Environment ou ambiente de execução Java)

- o JDK traz além JVM e das bibliotecas, o compilador

- javac (Java compiler)

- é o compilador do Java que pega nosso codigo fonte e traduz para uma linguagem que a JVM entenda

- para que possamos chamar o javac de qualquer diretório através de linha de comando precisamos editar o PATH do windows adicionando o caminho que se encontra o javac

- nesse caso estamos utilizando o seguinte caminho

C:\Program Files\Java\jdk-16.0.2\bin

- é correto afirmar em relação a JDK e JRE

- o JRE é o ambiente para executar uma aplicação Java

- se apenas quisermos executar uma aplicação Java basta o JRE

- o JDK além do ambiente de execução de aplicações Java, possui varias ferramentas de desenvolvimento

- ou seja o JDK são as ferramentas de desenvolvimento em si, mas também possui o JRE embutido

- JDK = JRE + ferramentas de desenvolvimento

- JVM – responsável por executar o bytecode

- a principal diferença entre a JVM e JRE é que:

- JRE (ambiente de execução) contem a JVM e muitas bibliotecas embutidas ou seja para rodar uma aplicação Java não basta somente a JVM, também é necessário as bibliotecas

- JRE = JVM + bibliotecas

- não baixamos somente a JVM

- baixamos a JRE que tem a JVM + bibliotecas

- a linguagem Java veio da C++

- é uma linguagem em que precisamos escrever um pouco mais (verbosa)

- a linha de comando que faz um print na tela é:

System.out.println("Ola mundo");

- todas instruções que damos sem chaves necessitam de ;

- todo codigo Java precisa estar dentro de uma classe

- a instrução System.out.println( )

- precisa estar dentro de um método chamado main

- é comum o uso de ***public*** antes de ***class Program*** embora não seja estritamente necessário mas utilizaremos

public class Programa {

        public static void main (String[] args) {

            System.out.println("Ola mundo");

        }

}

- o System.out.println ( ) poderá ser considerado um comando apesar de não ser um, mostrara algo na saída padrão (no nosso caso o CMD)

- o nome do arquivo é muito importante, precisa ser o mesmo nome da class

- no nosso caso ficara ***Programa.java***

\*\* no prompt do Windows temos o comando type (equivalente ao cat do terminal linux) que permite a visualização do conteúdo do arquivo

type Programa.java

- a extensão .java não é entendida pela JVM que entendera o bytecode com a extensão .class

- para compilar o nosso programa e transforma-lo em um bytecode utilizaremos o seguinte comando

javac Programa.java

- o comando acima transforma o Programa.java em Programa.class (binário = bytecode)

- para chamar a JVM utilizaremos

java Programa

- não utilizamos .class para chamar o programa

- se tentarmos chamar utilizando o comando

java Programa.class

- teremos um erro de execução pois o programa não se chama Programa.class e sim Programa (apesar de estar contido no arquivo Programa.class)

- erros comuns de compilação ocorrem quando esquecemos de utilizar o ; no final da instrução

- o Java possui palavras chaves (Keywords ou palavras reservadas), exemplo:

- public, class, static, void

- devem estar em letras minúsculas (o Java é case sensitive)

- chaves abrem e fecham blocos de códigos, exemplo

- tudo que encontra em ***public static void main***, pertence ao ***public class Programa***

- da mesma forma que ***System.out.println( )*** pertence a ***public static void main***

- são visíveis por meio das endentações

- no Java sempre a entrada de uma aplicação é a função (método) ***main***

***public static void main (String [] args)***

- sobre a compilação e execução do codigo Java:

- durante a compilação acontece uma verificação sintática do codigo

- tanto na compilação, quanto na execução podem aparecer erros

- JVM lê e executa o Bytecode

- o javac gera o Bytecode caso não tenha nenhum erro sintático no codigo fonte

- para compilar e executar um condigo Java utilizamos no exemplo

***javac Programa.java***

- para compilar passamos ao ***javac*** a extensão .java

***java Programa***

- para executar o programa passamos apenas o nome da classe sem a extensão .class

- para instalar a JDK no ubuntu

***sudo apt install default-jdk***

- para configurar o PATH do Windows

- digite editar variáveis de ambiente do sistema

- variáveis de sistema

- Path

- editar

- novo

- adicione o caminho para o diretório bin dentro da pasta jdk (que por sua vez esta dentro da pasta Java)

- salve as alterações e saia

- teste no prompt

***java -version***

***javac -version***

\*\* IMPORTANTE

- o nome do arquivo deve ser o mesmo nome da classe criada inclusive letras maiúsculas e minúsculas

- após a compilação com o javac o Bytecode é gerado com o mesmo nome, mas com .class no final

- um programa Java deve sempre estar escrito dentro de uma classe (class)

- toda instrução deve ser finalizada com ;

- {} abrem e fecham blocos de instruções

- um programa Java possui uma entrada que é uma função (método) main

System.out.println()

- imprime algo no console

03 – Começando com Eclipse

- o Eclipse é uma IDE (Integrated Development Enviroment 🡪 Ambiente Integrado de Desenvolvimento)

- diferente de um simples editor uma IDE integra em um único local:

- linguagem

- editor

- compilador

- biblioteca

- documentação

- era inicialmente um projeto codigo aberto que foi absorvido pela IBM e hoje é um consorcio de muitas empresas que tomam conta do projeto

- quando aberto o Eclipse pergunta sobre o workspace que nada mais é que o diretório onde teremos salvo os nossos projetos, podemos ter varias workspaces distintas

- o conjunto de janela dentro do Eclipse é chamado de perspectiva e cada janela é chamada de view

- cada view nos auxilia em uma situaçao

- para criarmos um projeto básico Java

🡪 File 🡪 New 🡪 Java project 🡪 nome do projeto 🡪 Finish

- após criado o projeto aparecera em Package explorer

- o diretório src (codigo fonte)

- JRE System Library (biblioteca com tudo que temos e acessível pelo Java)

- todos os comandos que utilizaremos estarão nesses arquivos .jar

- sobre o workspace

- cada proejto eclipse fica dentro de um workspace

- é uma pasta padrão que sera utilizada para armazenar todos os projetos criados com o eclipse

- para iniciar uma nova classe no Eclipse

- clicando em src com o botão direito 🡪 new 🡪 class 🡪 nomear a classe 🡪 finish

- codigo base do nosso programa que imprime Ola mundo!!!

***public class Programa {***

***public static void main (String [ ] args) {***

***System.out.println(“Olá Mundo!!!”);***

***}***

***}***

- na lateral esquerda a bolinha vermelha com x indica erro de compilação em determinadas linhas

- o \* ao lado do nome do arquivo significa que o mesmo ainda não foi salvo

- para executarmos o codigo através do Eclipse

***Run 🡪 Run As 🡪 Java Application***

- abre uma view para Console que abre executa o ***javaw.exe***

- que é uma versão do Java que não abre no Prompt do MS-DOS e é utilizado internamente pelo Eclipse para chamar nosso Programa

- o Package Explorer que é uma view esconde alguns arquivos e diretórios

- acessando o ***Window 🡪 Show View 🡪 Navigator***

- teremos acesso ao conteúdo do ***File System***

- dentro de Src no navigator teremos entre outros o ***Programa.class***

- clicando em ***src 🡪 Programa.java*** com o botão direito do mouse em properties teremos a localização do arquivo no nosso PC

- é correto afirmar:

- executamos nossos programas no Eclipse através do menu Run 🡪 Run as 🡪 Java Application

- dentro de um projeto Java, criamos uma classe através da opção do menu File 🡪 New 🡪 Class

- caso não seja um projeto Java a opção Class não estará disponível

- a saída do programa executado pelo Eclipse é feito pela view console

- a view navigator é parecida com o Windows Explorer ou Finder do MAC

- permite ver os diretórios do projeto com arquivos na integra

04 – Tipos e variáveis

- tipo inteiro

- criaremos uma nova classe para trabalhar com variáveis

- botão direito em src 🡪 New 🡪 Class 🡪 nome da nova classe

- no Java um statement (instrução) não funciona fora dos métodos, portanto precisaremos declarar

***public static void main(String [] args) {***

***}***

- dentro do método acima declaramos

***System.out.println(“Olá Mundo”);***

- para rodar nossa aplicação

***Run 🡪 Run as 🡪 Java Application***

***ou***

botão direito na classe main

***Run as 🡪 Java Application***

- as palavras que estão em roxo no Eclipse são palavras chaves, reservadas e deverão sempre estar em minúsculo

- o Java é uma linguagem estaticamente ou fortemente tipada

- necessita que declaremos todas variáveis e tipos das mesmas

- variáveis do tipo ***int***

- inteiros, sem casas decimais ou ponto flutuante

- int vem de integer

- então para declarar a variável idade por exemplo:

***int idade;***

***idade = 40;***

- primeiro dizemos o tipo que é, depois o nome, fechamos com ponto-e-vírgula e na linha seguinte inicializamos a variável idade atribuindo (=) o valor de 40

- o sinal = significa que estamos atribuindo algo

- podemos também criar a variável e inicializando-a na mesma linha

***int idade = 40;***

- para que a variável seja impressa no consolo temos que:

***System.out.println(idade);***

- não utilizamos aspas duplas, nesse caso pois queremos a variável em si (evaluation 🡪 resultado da expressão) e não uma cadeia de caracteres (string)

- quando passamos o mouse em cima da variável idade o Eclipse grifa (highlight) todas as ocorrências de idade, indicando que se tratam da mesma variável

- podemos utilizar operadores aritméticos na variável

- o Java segue a norma de precedência das operações, mas podemos utilizar os parênteses para determinar a ordem que queremos que elas sejam executadas

- algumas convenções de escrita de codigo no Java

- quando criamos alguma classe exemplo

***TestaVariaveis***

- a primeira letra da primeira palavra é maiúscula e a primeira letra das segunda palavra também, e assim por diante esse padrão de escrita chama-se camel case

- agora para iniciar o nome de uma variável utilizamos letra minúscula exemplo

***idade***

- caso a variável tenha duas palavras a segunda palavra inicia-se com letra maiúscula

***maioridade***

- não há costuma de abreviar palavras no Java

- podemos concatenar Strings com variáveis utilizando o operador +

***System.out.println (“ A idade de fulano é: “ + idade + “ anos.”);***

- ***ln***  do print pede ao Java para pular uma linha após sua execução

***System.out.print (“ A idade de fulano é: “ + idade + “ anos.”);***

- nesse segundo caso não haveria pulo de linha caso houvesse uma próxima instrução

- tipo flutuante:

- double

- quando tentamos utilizar um double em um tipo int teremos um erro de compilação

- “Type mismatch: cannot convert from double to int” 🡪 conversão não é possível

- existem dois tipos de variáveis para declarar um ponto flutuante, a mais utilizada é o double

- não é utilizado virgula para separar as casas decimais no Java e sim o ponto

39,0 🡪 errado

39.0 🡪 certo

- criaremos nesse exercício uma nova classe

🡪 default package 🡪 New 🡪 Class 🡪 nome da classe 🡪 Finish

***public class TestaPontoFlutuante {***

***public static void main (String [] args) {***

***double salario;***

***salario = 1500.90;***

***System.out.println(“Meu salario é “ + salario);***

***}***

}

- o resultado no consolo sera:

***Meu salario é 1500.9***

- o numero zero não aparece pois é suprimido após ser convertido para se unir a string

- podemos formatar conforme nossa necessidade através do formatters do Java (biblioteca) 🡪 não sera abordado neste curso

- podemos armazenar um numero do tipo inteiro (int) em uma variável do tipo ponto flutuante (double), porem o inverso não é possível

- uma das vantagens da rigidez do Java é a padronização quanto ao codigo quando se trabalhe em equipe

- podemos utilizar operações matematicas com o double

***double divisao = 3.14/2;***

***System.out.println(divisao);***

***1.57***

- se fizermos a seguinte divisao

***int outraDivisao = 5 / 2;***

- o resultado esperado seria 2.5 que não caberia em um tipo int

- porem no Java há uma regra que quando há uma divisao entre 2 numeros inteiros ele força um inteiro como resultado, logo o resultado da divisao acima será 2

- se tentarmos outra maneira de dividir

***double outraDivisao = 5 / 2;***

- o resultado do System.out.println(outraDivisao); sera 2.0

- isso ocorre porque porque o Java vai ler apenas o lado que vem antes da atribuição na linha de codigo relativa a double, ou seja primeiro realizara a divisao de dois números inteiros forçando um resultado inteiro e esse resultado o Java vai colocar na variável outradivisao mesmo que ela seja double o resultado armazenado pelo Java já é um int

- neste caso poderíamos utilizar do artificio de

***double outraDivisao = 5.0 / 2;***

- desta forma trata-se de um double dividido por um int e então a conta é feita levando em consideração o ponto flutuante esse ultimo exemplo trará um 2.5 como resultado

- ao declararmos uma variável é comum inicializa-la na mesma linha

***double salario;***

***salario = 1500.40;***

***double salario = 1500.40;***

- podemos realizar operações aritméticas misturando int com double por exemplo (10 / 3.14)

- conversões e outros tipos

- a conversão de um tipo int para um double é possível e chamado academicamente de promoção e ocorre de maneira automática

- para converter (fazer com que um double se encaixe em um tipo int) utilizaremos o casting) transforma o double em int

- exemplo abaixo:

***public class {***

***public static void main(String [] args) {***

***double salario = 1270.50;***

***int valor = (int) salario;***

***System.out.println(valor);***

***}***

***}***

- o valor printado no console sera 1270 (isso é chamado de casting)

- é necessário colocar após = o int entre parênteses para que a conversão ocorra

- no int cabem 32 bits com sinais (positivos e negativos) ou seja 2 elevado a 31 negativo e 2 elevado a 31 positivo (devido ao numero zero fazer parte da contagem)

- armazena cerca de 2 bilhoes, quando o numero ultrapassa esse valor teremos um overflow

- no long cabem 64 bits 2 elevado 63 – 1

- precisa de um l ou L no final para indicar que queremos um numérico long

203040590L ou 203040590l

- por padrão quando não é um double um numero no Java é considerado um int, o L (l) indica literal

- short 🡪 16 bits – 1

- byte 🡪 8 bits – 1

(-128 a 127)

- números maiores que 64 bits utilizam de objetos para serem declarados

- para lidar com dinheiro sem aparecer centavos utilizamos o BigDecimal

- os quatros tipos primitivos são

- int

- long

- short

- byte

- os dois tipos flutuantes são:

- double 🡪 64 bits

- float 🡪 32 bits

- para realizar um casting de um double para um float coloca-se no final F ou f

- veja os casting explícitos e implícitos na tabela abaixo

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

\*\* iniciamos nomes de classe SEMPRE em letras maiúsculas

\*\* funções e variáveis SEMPRE iniciamos em letras minúsculas

05 – trabalhando com caracteres

- char e string

- char guarda um único caractere de 16 bits

- o caractere atributos a um tipo char deve obrigatoriamente vir dentro de aspas simples

- armazena um único numero referente a tabela UNICODE

- se tentarmos realizar a seguinte operação

***valor = valor + 1;***

***System.out.println(valor);***

- teremos um erro de compilação em valor + 1 devido a ser dois tipos distintos em uma mesma operação e como o numero 1 é um int (maior que o tipo char) o Java não consegue realizar de forma implícita a conversão

- temos que realizar o type casting

***valor = (char) (valor + 1);***

***System.out.println(valor);***

- colocamos dentro dos parênteses valor + 1 pois queremos que tudo seja convertido

- String

- é escrito com letra maiúscula

- não é uma palavra reservada do Java, não guarda valor,

- é um tipo referencia

- para funcionar é necessário aspas duplas que podem inclusive ficar vazias o seu conteúdo

- no char as aspas simples não podem ficar vazias, temos que dar nem que seja um espaço

***String palavra = “Rodrigo Sundfeld”;***

***System.out.println(palavra);***

- podemos utilizar o operador + para concatenar a String

***String palavra = “Rodrigo Sundfeld” + 2021;***

***System.out.println(palavra);***

- saída no console sera

***Rodrigo Sundfeld2021***

Exemplo:

***String parcela1 = “10”;***

***String parcela2 = “20”;***

***System.out.println(parcela1 + parcela2);***

- o exemplo acima retorna no console ***1020***

- teremos a concatenação das duas Strings

- variáveis guardam valores

- um detalhe sobre todas a variáveis primitivas EXCETO String é seu funcionamento interno

- a tecla de atalho do Eclipse é Crtl + espaço após a palavra inicial ele nos dará um templates (autocomplete)

- dizemos que variáveis guardam valores e não uma referencia

**public** **class** TestaValores {

**public** **static** **void** main(String[] args) {

**int** primeiroNumero = 5;

**int** segundoNumero = 7;

segundoNumero = primeiroNumero;

primeiroNumero = 10;

System.***out***.println(segundoNumero);

System.***out***.println(primeiroNumero);

System.***out***.println(primeiroNumero + segundoNumero);

}

}

- quando fazemos um atribuição no Java não se diz que uma variável segue sempre a outra e sim que estamos copiando e colando valores

- no exemplo acima o primeiroNumero = 10 não surtira efeito para o segundoNumero

06 – Praticando condicionais

- testes com IFs

\*\* CRTL + espaço após digitar sysout + ENTER autocompleta ***System.out.printl();***

\*\* no eclipse

source 🡪 format

- formata o codigo de maneira automática

- no Java o espaço, TAB e ENTER não tem papel fundamental são utilizado para maior legibilidade do codigo

- quando temos apenas uma instrução dentro do IF ou do ELSE o uso de chaves {} não se faz necessário

- porem uma boa pratica é mesmo com uma instrução apenas dentro do bloco o uso das chaves {}

- quando o codigo se estende demais na tela podemos apertar o ENTER que o Eclipse faz com que as Strings sejam separadas por aspas “ + (concatenando) automaticamente

- um duplo clique em cima de qualquer view no Eclipse máxima a visualização da mesma e posteriormente outro duplo clique devolve a view ao tamanho original

**public** **class** TestaCondicional {

**public** **static** **void** main(String[] args) {

System.***out***.println("Testando condicionais.");

**int** idade = 17;

**int** quantidadePessoas = 3;

**if** (idade >= 18) {

System.***out***.println("Você pode entrar pois tem mais de 18 anos.");

System.***out***.println("Seja bem vindo!");

} **else** {

**if** (quantidadePessoas >= 2) {

System.***out***.println("Apesar de menor de 18 anos, " + "você pode entrar pois esta acompanhado.");

} **else** {

System.***out***.println("Infelizmente você não pode entrar pois tem menos de 18 anos.");

}

}

}

}

- boolean condicionais

- não é uma boa pratica dentro do codigo muitos IF e ELSE encadeados isso é chamado de complexidade ciclomatica ou complexidade condicional

- no Java não temos os operadores OR ou AND

- para representar OR (ou) utilizamos o pipe ||

- para o OR basta que apenas umas das condições seja verdadeira

- para representar AND (e) utilizamos &&

- para o AND todas as condições devem serem verdadeiras

- o boolean é uma palavra chave reservada do Java que é um tipo de variável que aceita somente true ou false

- no Java o sinal = significa atribuição, quando atribuímos algum valor ou algo

- enquanto == realiza a comparação (igualdade)

- temos com certa frequencia no Java é a direita do boolean colocarmos uma expressão booleana ***idade >= 18 && acompanhado***

- podemos substituir por ***boolean acompanhado = quantidadePessoas >=2;***

- o que faz com que saibamos se a pessoa esta ou não acompanhada

- podemos também imprimir uma String com o resultado do boolean utilizando o operador +

- operadores lógicos booleanos devem ter tanto do seu lado esquerdo quanto do seu lado direito uma expressão booleana, exemplo:

***if (idade > 18 && idade < 65) {***

***}***

**public** **class** TestaCondicional3 {

**public** **static** **void** main(String[] args) {

System.***out***.println("Testando condicionais.");

**int** idade = 17;

**int** quantidadePessoas = 3;

**boolean** acompanhado = quantidadePessoas >= 2;

**if** (idade >= 18 && acompanhado) {

System.***out***.println("Seja bem vindo");

} **else** {

System.***out***.println("Infelizmente você nao pode entrar pois tem menos de 18 anos e está desacompanhado");

}

}

}

**public** **class** TestaCondicional2 {

**public** **static** **void** main(String[] args) {

System.***out***.println("Testando condicionais.");

**int** idade = 17;

**int** quantidadePessoas = 1;

**if** (idade >= 18 || quantidadePessoas >= 2) {

System.***out***.println("Seja bem vindo");

} **else** {

System.***out***.println("Infelizmente você nao pode entrar pois tem menos de 18 anos e está desacompanhado");

}

}

}

- escopo e inicialização de variáveis

- se tentarmos utilizar uma variável antes que sua declaração tenha sido feita teremos um erro a nível de compilação

- a variável passa a valer a partir de sua declaração e vale somente entre as chaves correspondentes { } ou seja as chaves que que abrem e fecham isso chamamos de escopo

- para cada escopo temos que inicializar o valor da variável para que ela possa valer

- no Java as variáveis do tipo local são temporárias e não possuem valor padrão sendo necessária sua inicialização sempre antes de sua impressão, operação e assim por diante.

**public** **class** TestaEscopo {

**public** **static** **void** main(String[] args) {

System.***out***.println("Testando Condicionais.");

**int** idade = 20;

**int** quantidadePessoas =3;

**boolean** acompanhado;

**if** (quantidadePessoas >= 2) {

acompanhado = **true**;

} **else** {

acompanhado = **false**;

}

**if** (idade >= 18 || acompanhado) {

System.***out***.println("Seja bem vindo.");

} **else** {

System.***out***.println("Infelizmente você não pode entrar.");

}

}

}

**public** **class** TesteIR {

**public** **static** **void** main(String[] args) {

**double** salario = 3300.00;

**if** (salario >= 1900 && salario <= 2800.00) {

System.***out***.println("A sua aliquota de imposto é de 7.5% e você pode deduzir até R$142,00");

}

**if** (salario >= 2800.01 && salario <= 3751.00) {

System.***out***.println("A sua aliquota de imposto é de 15% e você pode deduzir até R$350,00");

}

**if** (salario >= 3751.01 && salario <= 4664.00) {

System.***out***.println("A sua aliquota de imposto é de 22.5% e você pode deduzir até R$636,00");

}

}

}

\*\* comando switch

- quando temos um programa onde temos que utilizar muitos IFs na sua construção uma solução é utilizar o ***switch*** 🡪 nele temos todos os rumos que nosso programa pode tomar

- o comando break irá interromper a execução do programa assim que uma condição for true para que as demais condições não necessitem ser executadas

- se nenhuma condição for aceita o codigo do ***default*** é que sera executado

- o switch então é uma solução para IFs encadeados

**public** **class** TestaMes {

**public** **static** **void** main(String[] args) {

// int mes = 12;

**int** mes = 13;

// int mes = 5;

**switch** (mes) {

**case** 1:

System.***out***.println("O mês é Janeiro");

**break**;

**case** 2:

System.***out***.println("O mês é Fevereiro");

**break**;

**case** 3:

System.***out***.println("O mês é Março");

**break**;

**case** 4:

System.***out***.println("O mês é Abril");

**break**;

**case** 5:

System.***out***.println("O mês é Maio");

**break**;

**case** 6:

System.***out***.println("O mês é Junho");

**break**;

**case** 7:

System.***out***.println("O mês é Julho");

**break**;

**case** 8:

System.***out***.println("O mês é Agosto");

**break**;

**case** 9:

System.***out***.println("O mês é Setembro");

**break**;

**case** 10:

System.***out***.println("O mês é Outubro");

**break**;

**case** 11:

System.***out***.println("O mês é Novembro");

**break**;

**case** 12:

System.***out***.println("O mês é Dezembro");

**break**;

**case** 13:

System.***out***.println("Mês inexistente");

**break**;

}

}

}

\*\* atalho Eclipse

CTRL + SHIFT + F

- formata o codigo fonte

07 – Controlando fluxo com laços

- laços com while

- while é uma palavra chave reservada do Java

- recebe obrigatoriamente dentro do parênteses uma expressão booleana (true ou false)

- sempre preciso declarar e inicializar a variável utilizada no while pois não acontece de forma automática em condições temporarias

- quando operamos sobre a própria variável existe uma forma sucinta de realizar o incremento do que 🡪 ***contador = contador + 1:***

***contador +=1*** ou ***contador++*** ou ***++contador***

- na expressão condicional while é possível utilizarmos qualquer operador de comparação (<, <=, >, >=, ==. !=) e também qualquer operador logico (&& 🡪 and || 🡪 or)

- o while sempre precisara de uma expressão condicional que definira quando o laço deve ser interrompido

- essa expressão deve ser informada dentro dos parênteses do while e pode utilizar qualquer dos operadores de comparação

- escopo nos laços

- toda vez que o Java entra no while é criada uma nova variável devido ao escopo e quando voltamos ao próximo laço (iteração) ela assume o valor pre-definido inicialmente

- exemplo de como deixar o codigo mais enxuto:

***total = total + contador; 🡪 total += contador;***

- quando declaramos a variável dentro do bloco do while teremos um erro de compilação pois o Java tenta acessar para realizar a condição do while uma variável que nem foi declarada e inicializada, para corrigir isso devemos colocar fora do bloco do while (escopo) a variável inicial.

- laço com for

- tanto while quanto o for são estruturas de laço e realizam a mesma tarefa

- o ***for*** proporciona uma maior legibilidade em alguns casos do codigo

- diferente do ***while*** no ***for*** não é preciso declarar a variável fora do escopo, devido a sua sintaxe

- utilizamos ; dentro dos parênteses do ***for*** para determinar seus parâmetros

- o ***for*** possui 3 parametros que são delimitados pelos parênteses

- primeiro parâmetro (opcional) que é a declaração e inicialização da variável

- este é executado apenas uma vez (no inicio do ***for***)

- o segundo parâmetro é executados em todas as vezes e contem a contem a condição booleana para determinar se entra o não no laço ou seja se executa a próxima iteração

- o terceiro parâmetro é ocupado pelo que deve ser feito ao final de cada iteração

- diferente do while apesar da variável valer no escopo do ***for*** durante suas iterações ele não é zerado, sendo executado apenas uma vez

- se quisermos realizar um **System.out.println();** do ultimo valor que o laço ***for*** estava lendo, *fora do* **fo***r não conseguiremos pois a variável* se resume ao escopo interno, ou seja deixa de existir depois do mesmo

**public** **class** TranformandoForEmWhile {

**public** **static** **void** main(String[] args) {

**for** (**int** contador = 0; contador <= 10; contador++) {

System.***out***.println(contador);

}

}

}

**public** **class** TestaWhile {

**public** **static** **void** main(String[] args) {

**int** contador = 0;

**while**(contador <= 10 ) {

System.***out***.println(contador);

contador++;

}

}

}

**public** **class** TestaFor {

**public** **static** **void** main(String[] args) {

**for**(**int** contador =0; contador <= 1000; contador++) {

System.***out***.println("O contador atual é: " + contador);

}

}

}

- laços encadeados

- laços que se unem a outros laços (aninhados)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

- mais laços com break

- break é uma palavra chave reservado do Java e tem como função parar a execução do laço mais interno

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**public** **class** TestaLacos2 {

**public** **static** **void** main(String[] args) {

**for** (**int** linha = 0; linha < 10; linha++) {

**for** (**int** coluna = 0; coluna < 10; coluna++) {

**if** (coluna > linha) {

**break**;

}

System.***out***.print("@");

}

System.***out***.println();

}

}

}

**public** **class** TestaLacos2 {

**public** **static** **void** main(String[] args) {

**for** (**int** linha = 0; linha < 10; linha++) {

**for** (**int** coluna = 0; coluna <= linha; coluna++) {

System.***out***.print("@");

}

System.***out***.println();

}

}

}

- exercício múltiplos de 3

\*\* o operador % chama-se modulo (resto do resultado de uma divisao)

/\*public class MultiplosDeTres {

public static void main(String[] args) {

for (int i = 1; i < 100; i++) {

if(i % 3 == 0) {

System.out.println(i);

}

}

}

}\*/

**public** **class** MultiplosDeTres {

**public** **static** **void** main(String[] args) {

**for** (**int** i = 3; i < 100; i += 3) {

System.***out***.println(i);

}

}

}