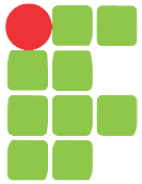


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Programação 1

Prof. Christiano Lima Santos

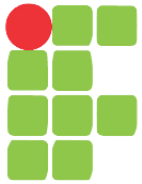
Objetivos

Espera-se que cada aprendiz alcance os seguintes objetivos:

- Compreender a lógica de programação e seus principais conceitos (variáveis, operações, atribuições, condições repetições, vetores e funções);
- Compreender os comandos/instruções básicas na linguagem de programação Java;
- Ser capaz de criar algoritmos/programas para tarefas simples.

Conteúdo do Curso

- Introdução à Programação
- Identificadores, variáveis, constantes e tipos de dados
- Atribuições, operadores e expressões
- Diretivas *package* e *import*, comentários, blocos de código e palavras reservadas
- Comandos para leitura e impressão
- Comandos condicionais
- Comandos de repetição
- *Strings* (textos)
- *Arrays* (vetores)
- Funções



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Introdução à Programação

Parte 01

Sumário

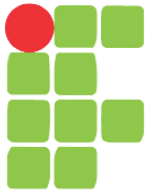
- Introdução à Programação
- Introdução aos algoritmos
- Introdução à linguagem Java
- Estrutura básica de um programa Java
- Ferramentas necessárias

Introdução à Programação

- Do ENIAC ao microcomputador, dos supercomputadores à computação ubíqua, a Computação passou por grandes evoluções e transformações;
- Quais as contribuições da Computação nas seguintes áreas?
  - Educação;
  - Saúde;
  - Finanças;
  - Política.

Introdução à Programação

- Você consegue imaginar um mundo sem a Computação?
- De forma similar, o mundo não seria o mesmo sem os avanços da programação dos computadores!



Mas... Programar é difícil?

- Possui os mesmos desafios de aprender algo muito diferente pela primeira vez!
  - Falar;
  - Ler e escrever;
  - Cálculos aritméticos.

- Aprender a programar exige praticar!

Sugestão
Dicas para aprender a programar

Sugestão de vídeo:

https://youtu.be/ZtMzB5CoekE

O que são algoritmos?

- São uma forma de descrever como uma determinada tarefa deve ser executada, de forma clara e objetiva;
- Podem ser usados para descrever como um *software* deve proceder para cumprir uma tarefa.

Características de um algoritmo

- **Finitude** – Deve ser formado por um conjunto finito de passos, isto é, possuir início e fim;
- **Definição** – Cada ação ou passo de um algoritmo deve ser escrito de forma bastante precisa, sem ambiguidades ou dúvidas quanto à sua execução;
- **Sequencial** – Um algoritmo é formado por uma série de passos que devem ser executados em uma ordem sequencial;

Características de um algoritmo

- **Entradas** – Deve apresentar zero ou mais entradas, isto é, informações a serem inseridas por um usuário ou por outra fonte externa;
- **Saídas** – Deve apresentar uma ou mais saídas, isto é, informações pré-definidas e/ou resultantes do processamento das entradas;
- **Eficiência** – Significa que suas operações devem ser suficientemente básicas tal que um ser humano seja hábil a executá-las com precisão em papel e lápis em um tempo finito.

Formas de representação dos algoritmos

- Descrição narrativa
- Fluxograma
- Pseudocódigo
- Em uma linguagem de programação

Descrição narrativa

- Descreve como uma tarefa deve ser executada por meio de um texto, seja ele verbal ou escrito;

Exemplo - Receita de bolo

Separar os ingredientes

Bater os ovos em ponto de neve na batedeira

Acrescentar açúcar e farinha de trigo

Acrescentar uma colher de manteiga

Acrescentar uma colher de fermento em pó

Colocar na forma

Colocar no forno e assar

Tirar da forma e servir

Fim do processo!

Descrição narrativa

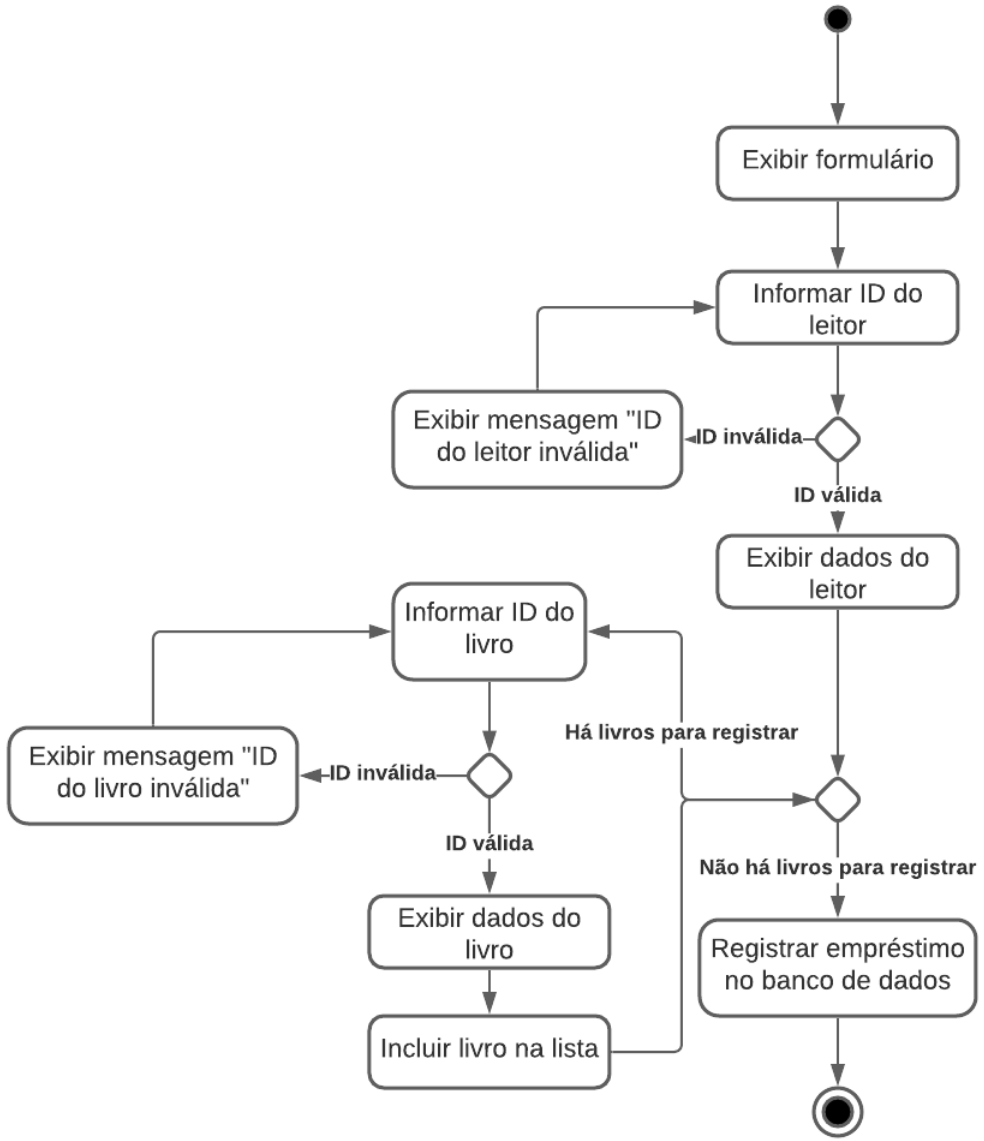
- **Vantagem:** muito fácil de ser compreendida e empregada no dia-a-dia, qualquer um capaz de compreender alguma língua pode falar ou escrever uma narrativa na mesma.
- **Desvantagem:** inexistência de regras que ditem como cada ação deve ser descrita a fim de evitar ambiguidades ou problemas de clareza quanto à sua compreensão.

Fluxograma

- Trata-se de um diagrama composto por várias ações e entidades representadas em “caixas” de diversos formatos (cada formato determina o tipo de ação a ser tomado ali) e setas interligando todas essas caixas de forma a criar-se um “caminho” com início e fim.
- A sequência de ações que deve ser tomada, então, é encontrada “percorrendo-se” o fluxograma de seu início até o seu fim, sendo então executada cada ação ou teste como está escrito.

Fluxograma

Exemplo - Efetuar empréstimo de livro



Fluxograma

- **Vantagem:** é mais concisa que a descrição narrativa, representando visualmente desvios e laços;
- **Desvantagem:** necessita o aprendizado do processo de construção de fluxogramas.

Pseudocódigo

- É a representação textual das ações por meio de comandos pré-definidos em uma espécie de linguagem similar a uma linguagem de programação, porém muito mais próxima da nossa própria;
- No Brasil, temos o Portugol ou Português Estruturado.

Exemplo - Cálculo da média de um aluno

```
programa calculo_de_media;
leia notaA, notaB, notaC;
media <- (notaA + notaB + notaC) / 3;
imprima media;
```

Pseudocódigo

- **Vantagem:** sua transcrição para uma linguagem de programação é bastante direta;
- **Desvantagem:** requer aprender todas as regras daquele pseudocódigo, o que pode ser quase tão complexo quanto aprender uma linguagem de programação.

Em uma linguagem de programação

- Pode-se utilizar de uma linguagem de programação tal que possamos elaborar diretamente o código-fonte do programa que desejamos.

Exemplo - Cálculo da média de um aluno

```
public void main() {
    float notaA = 7;
    float notaB = 8;
    float notaC = 9;
    float media = (notaA + notaB + notaC)/3;
    System.out.println("Média: " + media);
}
```

Em uma linguagem de programação

- **Vantagem:** por meio dela, já temos o código-fonte do programa que desejamos construir, assim, precisamos somente compilá-la ou interpretá-la;
- **Desvantagem:** é necessário conhecimentos suficientes de uma linguagem de programação a fim de escrever o código-fonte da mesma, exigindo assim muito mais do aprendiz.

Elementos fundamentais de um algoritmo

- Todo algoritmo possui:
  - Entrada;
  - Processamento;
  - Saída.

- Para ficar mais claro, consideremos como exemplo um algoritmo que, dado um número N, deve retornar o fatorial de N como solução.

Elementos fundamentais de um algoritmo

- **Entrada:** são os dados de entrada do algoritmo, informação necessária para a execução correta do mesmo e deve ser informada pelo usuário do sistema e/ou vir de outras fontes externas (como bancos de dados, arquivos etc.);

- No nosso exemplo, o valor de N será a nossa entrada, necessária para o cálculo.

Elementos fundamentais de um algoritmo

- **Processamento:** são os procedimentos utilizados para, a partir dos dados de entrada, chegar ao resultado final.

- Em nosso exemplo, todo o cálculo necessário para determinar o fatorial de N deve ser considerado como parte do processamento.

Elementos fundamentais de um algoritmo

- **Saída:** são os dados já processados e prontos para serem enviados ao usuário ou armazenados em alguma fonte externa;

- Em nosso exemplo, o resultado final, isto é, o fatorial de N, é a nossa saída.

Elementos fundamentais de um algoritmo

O que é uma linguagem de programação?

- Assim como estabelecemos uma linguagem para a comunicação entre duas ou mais pessoas, os computadores possuem sua própria linguagem;
  - Linguagem de máquina (ou linguagem de baixo nível);
- Entretanto, por serem linguagem difíceis para nós entendermos, foram criadas linguagens de programação mais parecidas com a linguagem humana, como C/C++, Java, PHP etc.
  - Linguagens de alto nível.

O que é uma linguagem de programação?

- Assim, uma linguagem de programação refere-se ao conjunto de regras sintáticas e semânticas utilizado para definir um programa de computador;

- Por meio dela, somos capazes de especificar os comandos necessários para a execução das tarefas em computador. Essa especificação é geralmente conhecida como código-fonte.

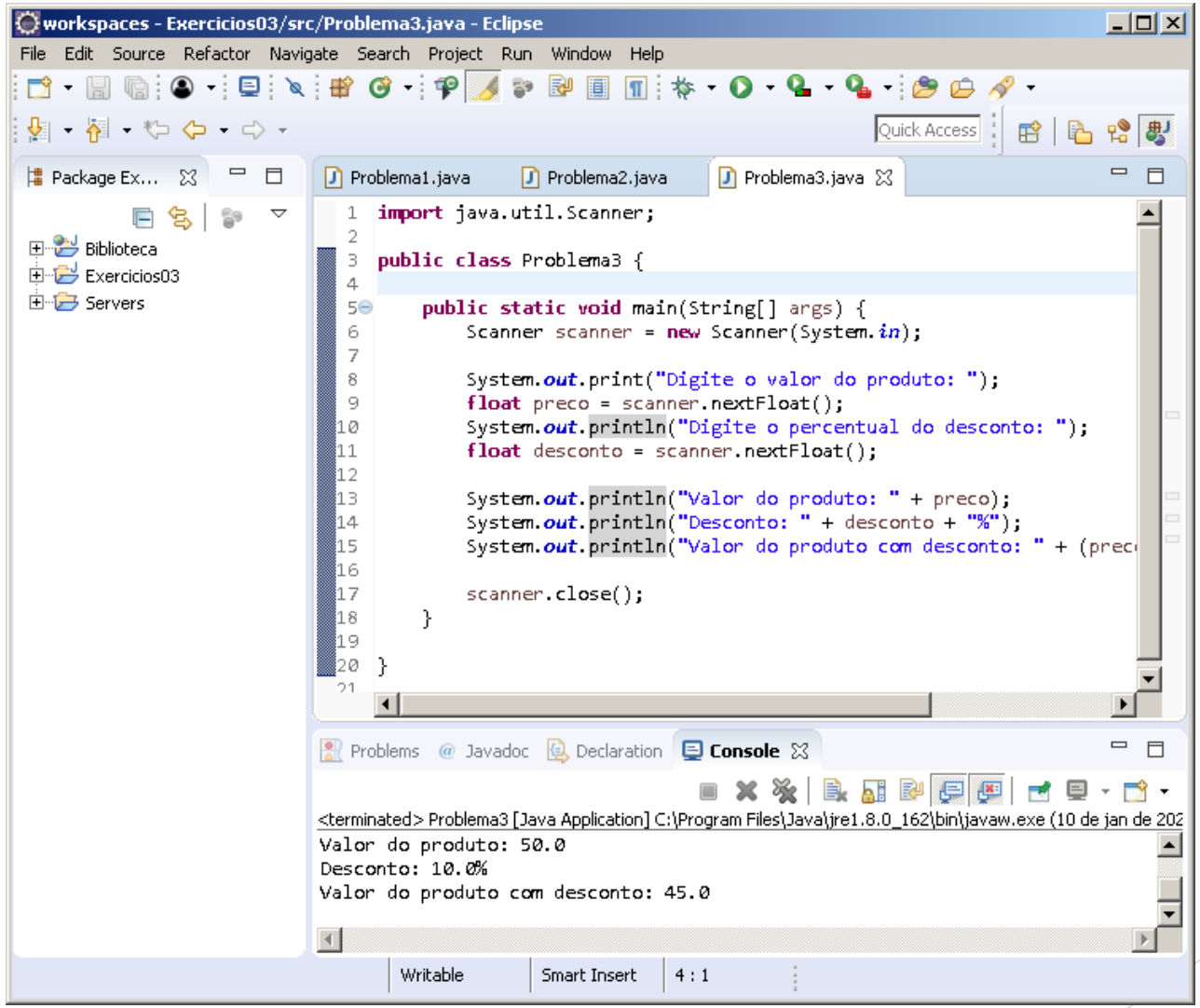
O que é uma linguagem de programação?

- Então, um *software* é responsável por converter o nosso código-fonte (linguagem de alto nível) em linguagem de máquina;
  - Compilação;
  - Interpretação;
  - Compilação para Máquinas Virtuais.

Integrated Development Environment (IDE)

- Refere-se a um ambiente integrado onde o desenvolvedor pode organizar todos os arquivos relacionados ao desenvolvimento de um programa, editá-los, compilá-los e testar seu programa de forma fácil e rápida;
- Antes do uso de IDEs, o código-fonte era escrito em qualquer editor de texto (sem benefícios como funcionalidades para “autocompletar o código”, verificação de sintaxe em tempo de edição e outras) e era comum o uso de *prompt* de linha de comando para as tarefas de compilação, linkedição, etc.

Integrated Development Environment (IDE)



IDE Eclipse

A plataforma Java

- Desenvolvida no início da década de 90 pela Sun Microsystems (posteriormente adquirida pela Oracle), a plataforma Java visava permitir a criação de aplicativos independentes de plataforma, baseado no ditado “*write once, run everywhere*" (escreva uma vez, execute em todo lugar);

- Inicialmente, essa plataforma permitia a execução de *applets* rodando dentro dos navegadores web. Tal solução não vingou, mas abriu espaço para outras propostas futuras;

A plataforma Java

- A plataforma consiste em três partes principais:
  - Linguagem de programação (Java);
  - Máquina virtual (JVM);
  - Diversas APIs.

- A linguagem Java é uma linguagem orientada a objetos. Desde sua introdução, em 1995, tornou-se uma das mais populares linguagens de programação do mundo.

A plataforma Java

- Para desenvolver nessa plataforma, são necessários:
  - O Kit para Desenvolvimento de Software (SDK) do Java;
    - JDK 8 update 162, 64bit https://christianosantos.com/files/p1/jdk_64bit_v8_u162.exe
  - Um Ambiente de Desenvolvimento Integrado (IDE) configurado.
    - Eclipse Oxygen (março 2018) https://christianosantos.com/files/p1/eclipse.zip

A plataforma Java

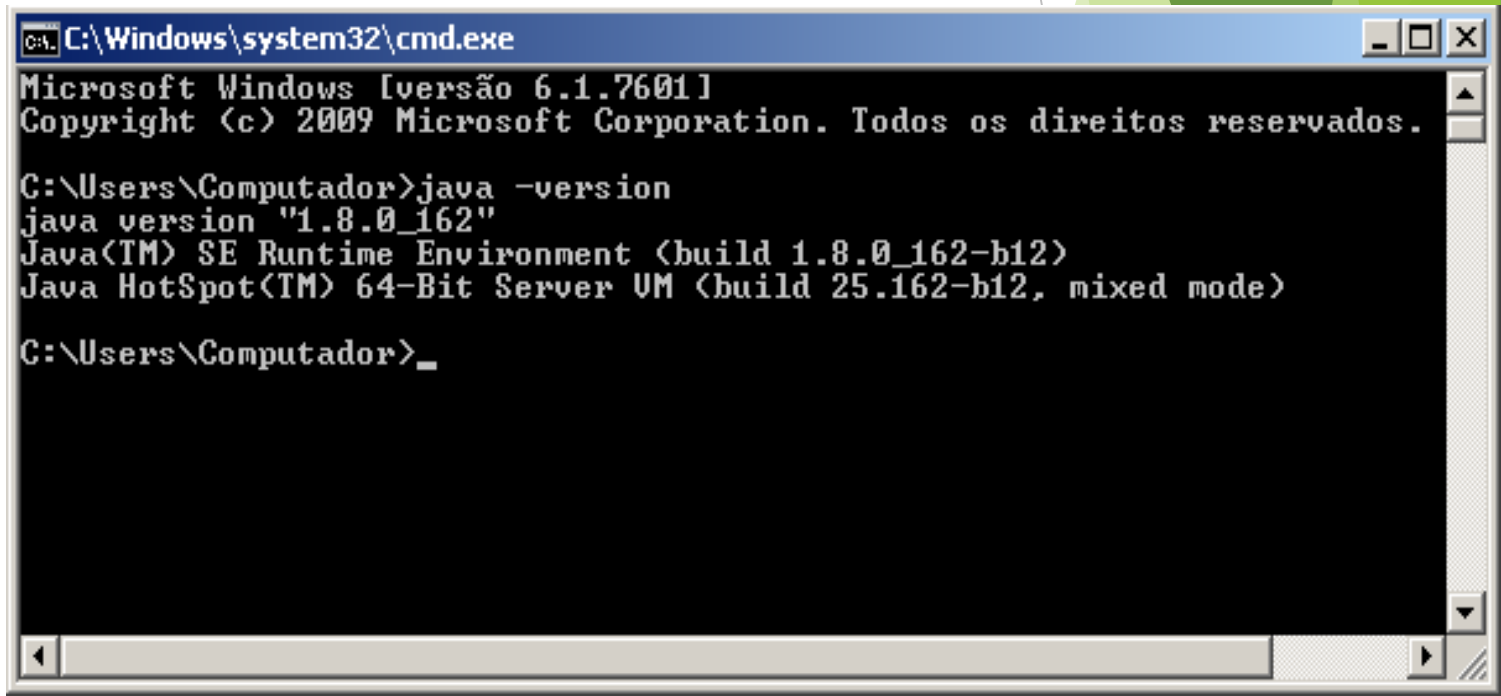
- **Obs:** Em Programação 3, será utilizado também um servidor Java para web;
  - No arquivo do Eclipse, na pasta “Servers”, encontra-se o Apache Tomcat 7.0.85.

Preparando o ambiente de desenvolvimento

- Após baixar e instalar o JDK em seu PC, verifique se o mesmo foi instalado corretamente, abrindo o prompt e digitando a seguinte linha de comando:

  java -version

- Se o sistema responder informando a versão do Java, então foi instalado corretamente.

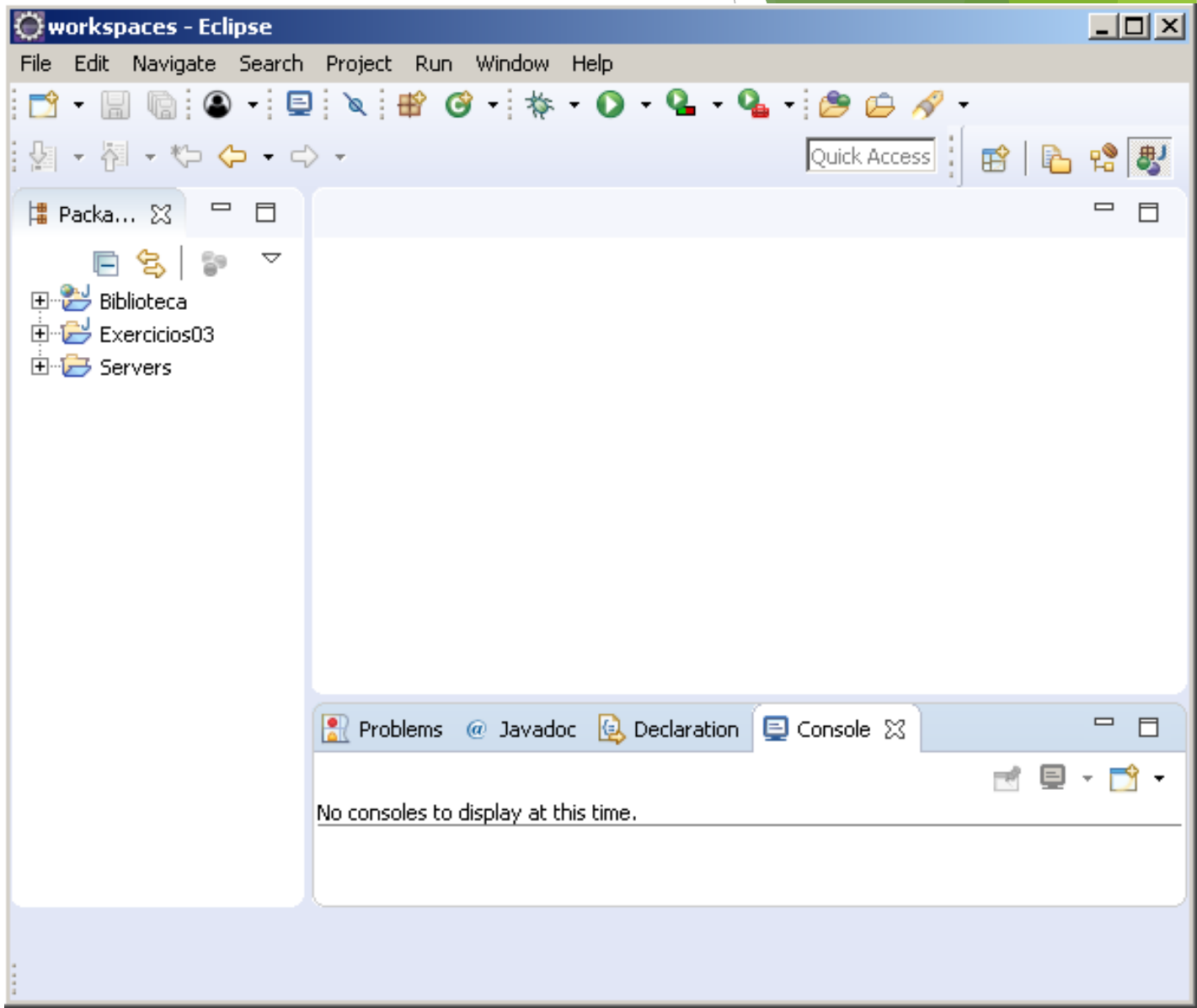


Preparando o ambiente de desenvolvimento

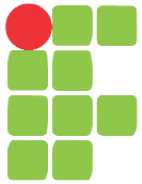
- Próximo passo, baixe e descompacte o Eclipse em seu computador (pode ser na raiz da unidade C: ou outra);
- Abra a pasta, procure pelo arquivo executável “eclipse.exe” e dê um duplo clique;
- Como é a primeira vez, o Eclipse vai solicitar que indique ou crie uma pasta em seu computador para servir de *workspace*. Crie uma pasta ou indique uma já existente;
  - Um *workspace* é uma área de trabalho, uma pasta onde ficam armazenados vários projetos.

Preparando o ambiente de desenvolvimento

- Após escolha da pasta para workspace, o Eclipse abrirá sua janela!



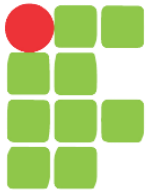
Exercícios



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Identificadores, Tipos de Dados, Variáveis e Constantes

Parte 02



Sumário

- Identificadores
- Tipos de Dados
- Variáveis
- Constantes

Recordando...

- O que é programar?
- O que é um algoritmo?
- O que é uma linguagem de programação?

Identificadores

- Um identificador refere-se ao nome dado a uma variável, tipo, função, constante etc.
- Java é uma linguagem *case-sensitive*, isto é, que diferencia letras maiúsculas de minúsculas, então Objeto, objeto e OBJETO são identificadores diferentes e podem se referir a entidades (variáveis, constantes, funções etc.) diferentes dentro de um programa.

Regras para criação de identificadores

- Um identificador deve conter somente letras (maiúsculas ou minúsculas), dígitos ou um grupo restrito de caracteres especiais (_, $, £, ¢);
  - **Obs:** Em Java, não é muito comum criarmos variáveis iniciando com "_", muito menos usando os outros caracteres especiais ($, £, ¢);
- Não podem ser utilizados outros caracteres especiais, como letras com acento, "&" ou o espaço;
- O primeiro caracter de um identificador não pode ser um número;

Regras para criação de identificadores

- Letras maiúsculas e minúsculas podem ser utilizadas, mas é uma boa prática de programação:
  - Em “não-constantes” (variáveis, funções etc.) – utilizar sempre letras minúsculas, excetuando a primeira letra da segunda palavra em diante que compõe o identificador. Exemplo: somaTotal, pagamentoDosEmpregados;
  - Em constantes – utilizar sempre letras maiúsculas e empregar o “_” entre palavras diferentes compondo o identificador. Exemplo: SOMA_TOTAL, PAGAMENTO_DOS_EMPREGADOS;
  - Em classes - iniciar por letras maiúsculas. Exemplo: Usuario, Livro, Math;
  - Em pacotes - iniciar por letras minúsculas. Exemplo: br.edu.ifs.calendario

Exemplos de identificadores

Identificadores válidos

a

A

_a

$usuario

a1

a_1

casa1

somaTotal

valor_parcela

_salarioBase

Identificadores inválidos

1a

casa 1

calçada

soma total

Salário Base

3nota

Obs: o identificador "_" é válido e pode até ser usado, porém Java recomenda que não, por ser palavra reservada na linguagem

Tipos de dados

- Um tipo especifica as características de uma informação, determinando assim quais valores podem ser armazenados, como podem ser lidos ou escritos e quais as operações possíveis com a mesma;
- O tipo determina também o intervalo de valores possíveis. Por exemplo, existem tipos em que “cabem” números maiores do que em outros.

Tipos mais comuns em Java

Tipo	Valores aceitos	Exemplos de valores
int	inteiros	-3 0 7 56
float	decimais	0.5 -2.3 10.0 12
boolean	valores lógicos	true (verdadeiro) false (falso)
char	caracteres	'@' 'A' '6'
String	textos	"A Santa Casa" "moeda" "c" ""

Observações:

Perceba que o tipo String é escrito com letra inicial maiúscula. Na verdade, ele é uma classe (falaremos sobre isso mais adiante no curso).

Existe um tipo especial chamado “void”, usado em métodos para dizer que eles não retornam (devolvem) nenhum valor.

Variáveis

- Uma variável trata-se de um local na memória reservado para armazenar valores de um determinado tipo. Esse local na memória poderá ser referenciado (isto é, acessado para leitura ou escrita) a partir do identificador dado àquela variável;
- Vale lembrar que o valor armazenado por uma variável **pode ser alterado durante a execução** do programa, principal distinção entre uma variável e uma constante!

Declaração de variáveis

- Em C, nós precisamos declarar as variáveis antes de usá-las. Abaixo se encontra a especificação de como deve ser feita a declaração de uma variável:

<tipo> <identificador> [= <constante>] [,<identificador> [= <constante>]];

Exemplo:

int idade = 5;

Exemplos de declaração de variáveis

Declarações válidas

```
int a = 5;
float media = 6.5;
int i, j;
String nome = "Carlos";
boolean ativo = false;
float nota1 = 5, nota2 = 7;
```

Declarações inválidas

```
Float b = 3;
float nota = "6.4";
boolean x y;
String escola = IFS;
boolean presente = TRUE;
float media = 5,7;
```

Constantes

- Uma constante trata-se de uma referência (nome) para um dado valor, permitindo sua leitura (mas não escrita) a qualquer momento;
- É importante salientar que uma constante é, então, um valor (referenciado por um identificador) que **não pode ser alterado durante a execução** do programa!

Declaração de constantes

- Em Java, nós também precisamos declarar as constantes antes de usá-las. Abaixo se encontra a especificação de como deve ser feita a declaração de uma constante:

final <tipo> <identificador> [= <valor>] [,<identificador> [= <valor>]];

Exemplo:

final int c = 3;

Exemplos de declaração de constantes

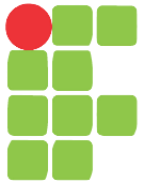
Declarações válidas

```
final int a = 5;
final float media = 6.5;
final int i, j;
final String nome = "Carlos";
final boolean ativo = false;
final float nota1 = 5, nota2 = 7;
```

Declarações inválidas

```
Final float b = 3;
final float nota = "6.4";
final boolean x y;
final String escola = IFS;
final boolean presente = TRUE;
final float media = 5,7;
```

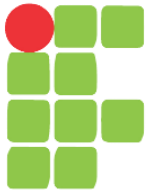
Exercícios



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Operações e Expressões

Parte 03



Sumário

- Operadores, Operandos e Expressões
- Tipos de operadores em Java

Recordando...

- Quais os tipos de dados mais comuns em Java?
- O que são variáveis?
- Como você declararia três variáveis para guardar as notas 5, 7.5 e 8.4?
- O que são constantes?
- Qual a diferença entre variáveis e constantes?

Algumas definições: Operadores, Operandos e Expressões

- Um **operador** é um símbolo utilizado para identificar que uma determinada operação deve ser realizada sobre um ou mais parâmetros, denominados **operandos**;

Exemplos:

Em “3 + 4”, o “+” é o operador, atuando sobre os operandos “3” e “4”.

Em “x + (3 * 5)”, temos o operador “+” atuando sobre os operandos “x” e “3 * 5”, e o operador “*” atuando sobre os operandos “3” e “5”.

Algumas definições: Operadores, Operandos e Expressões

- Já uma **expressão** pode ser considerada um conjunto de operações efetuadas sobre certos dados a fim de obter um resultado;

Exemplos:

“3 + 4” é um exemplo de expressão;

“x * 5 / y”, também é um exemplo de expressão;

“7 > x”, também!

Tipos de operadores

- Operadores aritméticos
- Operadores relacionais
- Operadores lógicos
- Operadores de atribuição

Operadores aritméticos

Operador	Significado	Exemplo
+	Soma	5 + 2
++	Incrementa uma unidade	i++
-	Subtração ou inversão de sinal	5 - 2 -3
--	Decrementa uma unidade	i--
*	Multiplicação	5 * 2
/	Divisão	5 / 2
%	Módulo (resto)	5 % 2

Operadores relacionais

Operador	Significado	Exemplo
>	Maior	5 > 2
<	Menor	5 < 2
>=	Maior ou igual	5 >= 2
<=	Menor ou igual	5 <= 2
==	Igual	5 == 5 5 == "5"
!=	Diferente	5 != 2

Operadores lógicos

Operador	Significado	Exemplo
&&	E	x && y
||	Ou	X || y
!	Não	!x

Operadores para strings

Operador	Significado	Exemplo
+	Concatenação	x + y

Operadores de atribuição

Operador	Significado	Exemplo
=	Recebe (armazena um valor)	x = 3
+=	Recebe o valor dele adicionado com	x += 3
-=	Recebe o valor dele subtraído de	x -= 3
*=	Recebe o valor dele multiplicado por	x *= 3
/=	Recebe o valor dele dividido por	x /= 3
%=	Recebe o resto da divisão dele por	x %= 3

Obs: Na verdade, todo operador binário pode ser usado junto com a atribuição, tornando-se uma **atribuição composta**.

Exercícios

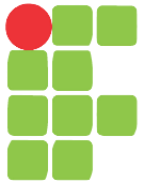
- Declare duas variáveis inteiras **a** (valor inicial 5) e **b** (valor inicial 7). A seguir, declare uma variável decimal **c** e atribua a ela o resultado da multiplicação de **a** por **b**;

- Declare duas variáveis decimais **x** (inicialmente 3.5) e **y** (inicialmente 5 vezes o valor de **x**). Declare, então, uma variável decimal **z** e atribua a esta a metade do quadrado da soma de **x** e **y**.

Exercícios

- Escreva as seguintes atribuições em Java:

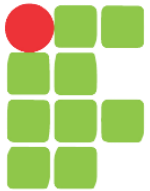
$$a = \quad 1.0$$
$$b = \quad -1.0$$
$$c = -12.0$$
$$delta = \quad b^2 - 4ac$$
$$x1 = \frac{-b - \sqrt{delta}}{2a}$$
$$x2 = \frac{-b + \sqrt{delta}}{2a}$$



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Diretivas, comentários, blocos de código e palavras reservadas

Parte 04



Sumário

- Diretivas
  - *package*
  - *import*
- Comentários
- Blocos de código
- Palavras reservadas

Recordando...

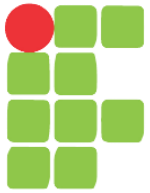
- Quais as formas disponíveis para incrementar o valor de uma variável em 1?
- Considerando duas variáveis **a** e **b**, como proceder para trocar o valor delas?
- Como podemos calcular o resto da divisão de um número por outro?
- Como podemos calcular e armazenar em uma variável 10% do valor de outra?

Diretivas

- Diretivas são instruções/comandos presentes em muitas linguagens de programação que orientam o compilador ou montador sobre como processar o código-fonte;

- Assim, podem ser utilizadas para especificar classes usadas pelo código, a organização dos arquivos, definir *namespaces* etc.

- Geralmente são lidas e executadas antes da compilação numa etapa conhecida como pré-compilação ou pré-processamento.



Diretivas

- Java suporta diversas diretivas, sendo que duas das mais comuns são as diretivas *package* e *import*.

Diretiva *package*

- Esta diretiva especifica o pacote em que uma classe se encontra e geralmente é a primeira linha presente no arquivo da classe (exceto se não for especificado um pacote, adotando assim o "pacote padrão" – *default package*);

- Em Java, em nível de arquivos, um pacote trata-se de uma estrutura hierárquica de diretórios no qual classes pertencentes a um mesmo pacote estarão.

Diretiva *package*

- Algumas vantagens do uso de pacotes em um programa:
  - Melhor organização física do código de um projeto, agrupando classes com finalidades (interesses) similares em um mesmo pacote;
  - Evita colisão de classes com mesmo nome, porém pertencentes a pacotes diferentes;
  - Aplicativos móveis (Android ou iOS) são identificados por meio de um pacote.

Diretiva *package*

Pacote	Funcionalidade
java.awt	AWT, ou Abstract Window Toolkit. Contém todas as classes para a criação de aplicações com interfaces gráficas com o usuário (ou GUI, Graphical User Interface).
java.io	Fornece as classes para realizar operações de entrada (input) e saída (output) através do sistema de fluxos de dados (streams) e serialização de objetos.
java.lang	Fornece classes que são fundamentais para a concepção da linguagem de programação Java.
java.net	Fornece as classes para implementar aplicações de rede.
java.util	Para trabalhar com coleções, entrada de dados (Scanner) e componentes de data e hora.
javax.swing	O pacote Swing é um pacote de extensão que complementa o pacote AWT.

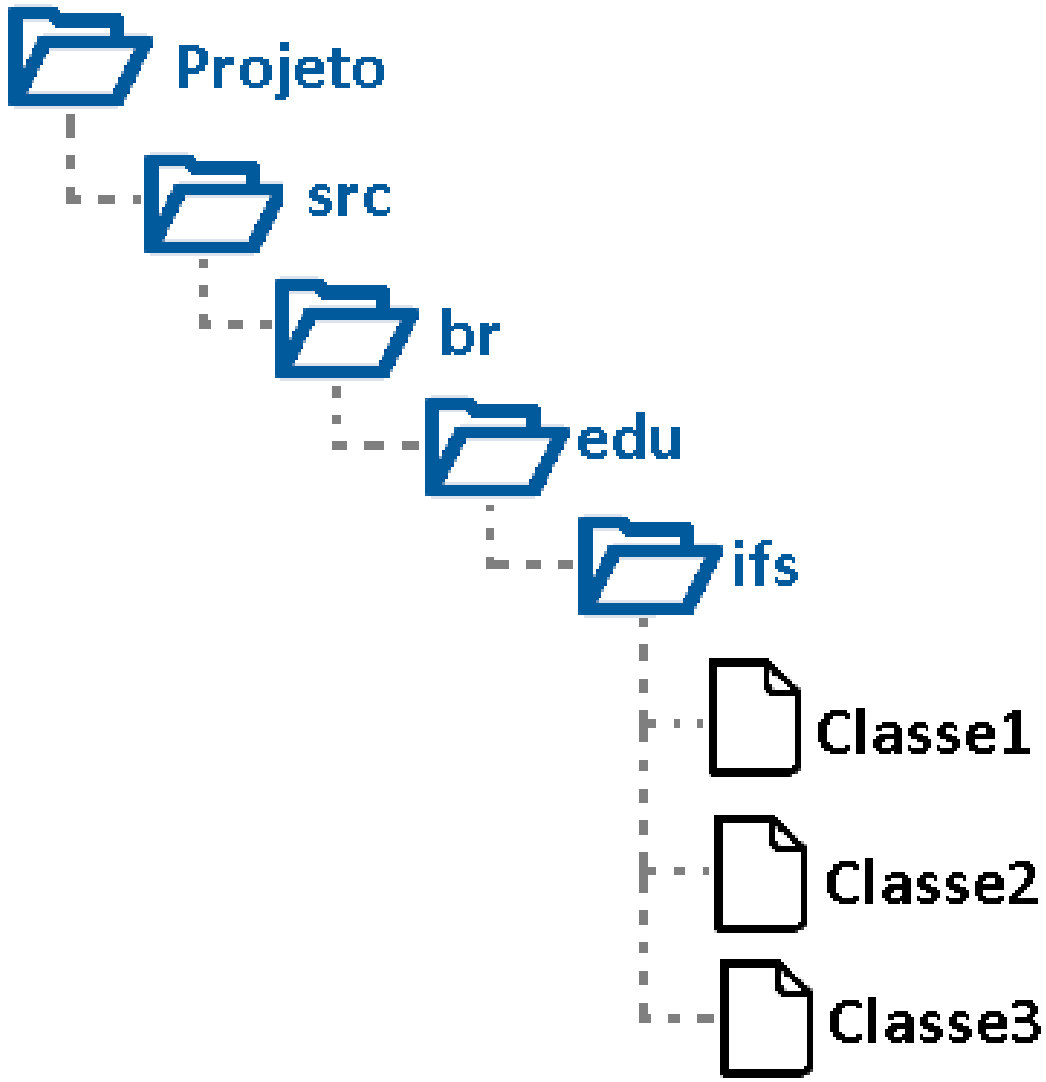
Pacotes mais comuns da API da plataforma Java

Fonte: https://www.devmedia.com.br/compreendendo-o-uso-de-pacotes-ou-packages/27478

Diretiva *package*

- Nomes de arquivos geralmente são compostos por vários identificadores em letras minúsculas (cada identificador refere-se a uma pasta), unidos por meio de “.”;

- Em nível de arquivos, um pacote chamado “br.edu.ifs” trata-se de uma estrutura de pastas dentro da pasta do projeto (“src”), começando pela pasta “br”, que dentro terá a pasta “edu” e, dentro desta, a pasta “ifs”, onde ficarão nossas classes. Assim:

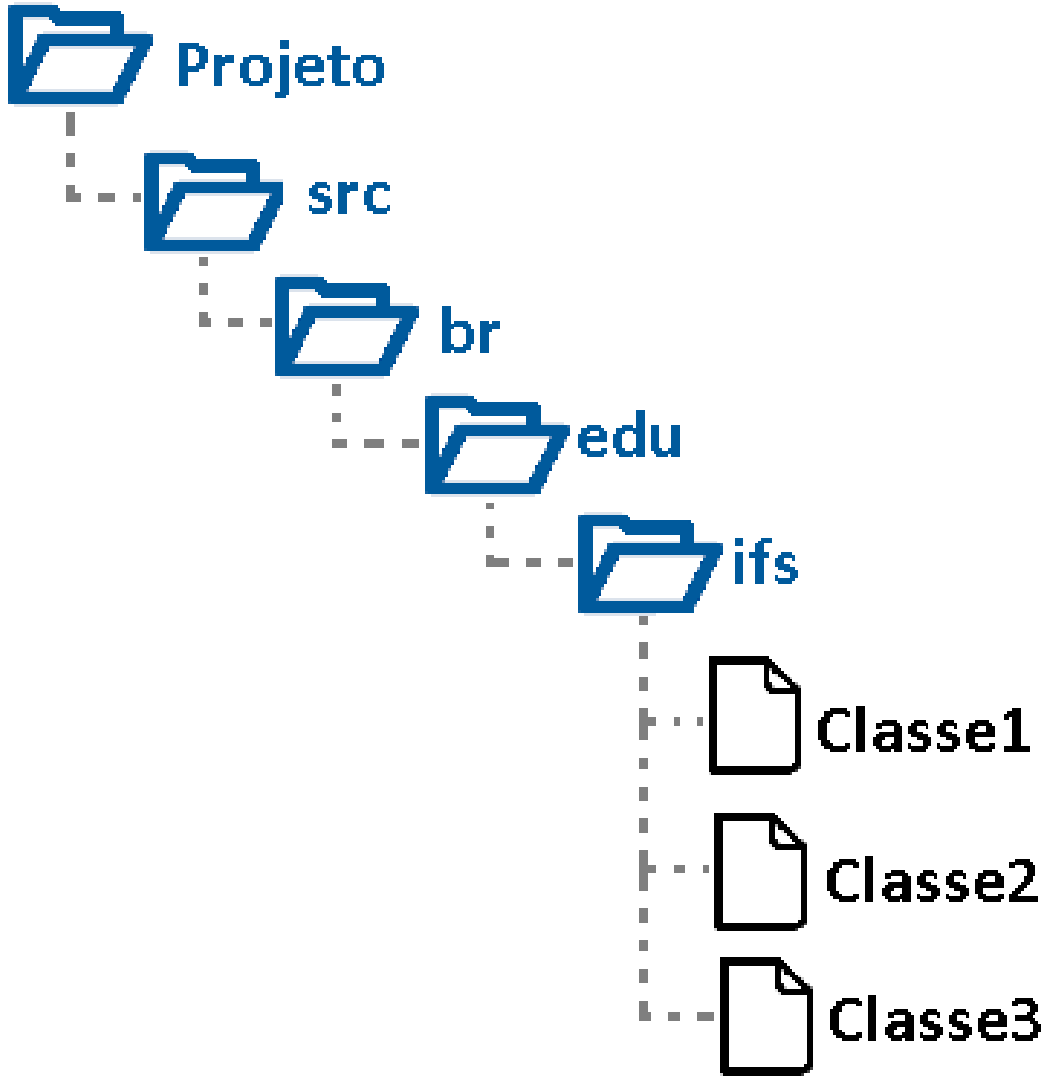


Diretiva *package*

- As classes do exemplo anterior terão, no início de seu código, a seguinte diretiva:

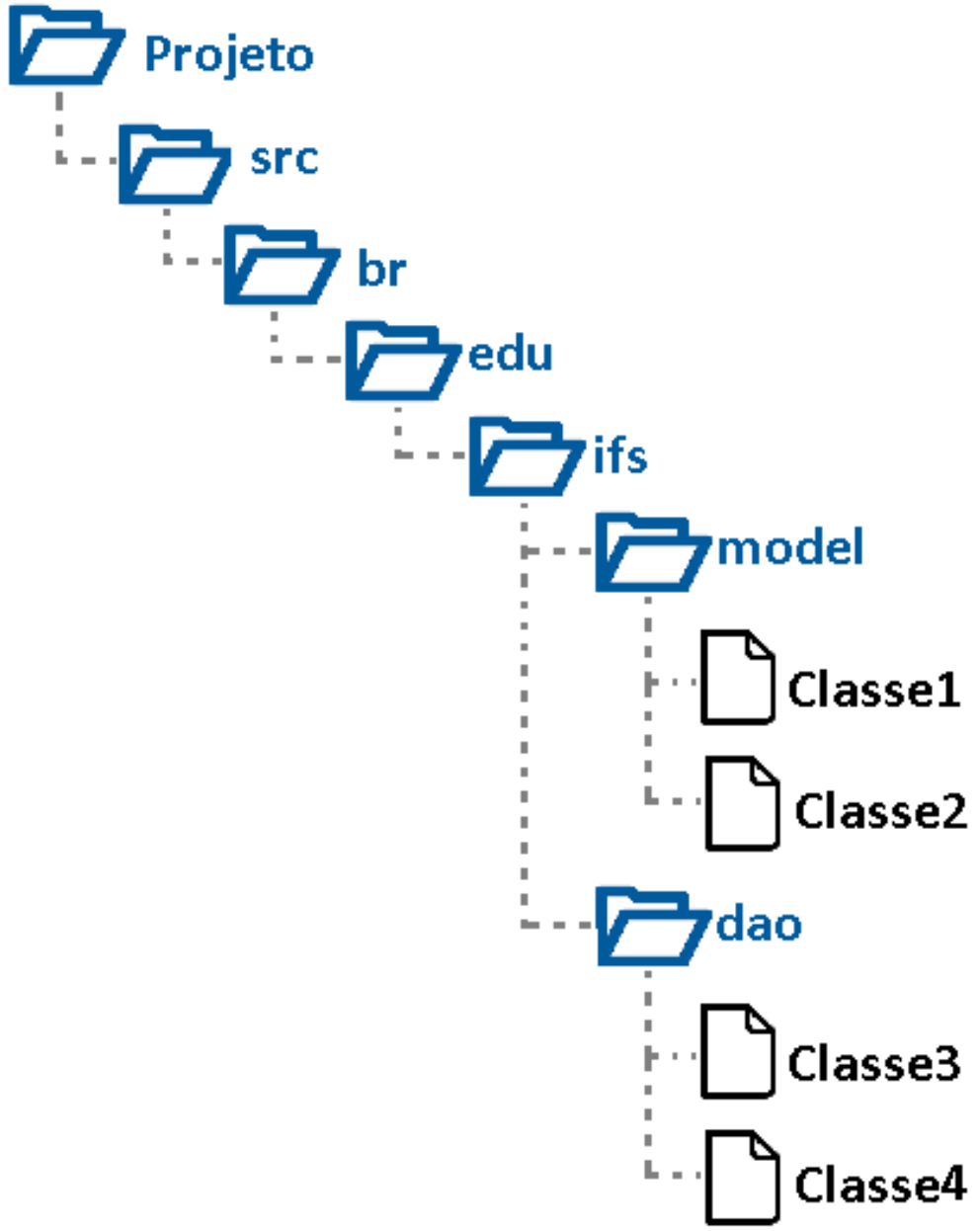
package br.edu.ifs;

Dica: pacotes podem ter nomes tão comuns como somente uma palavra (“lista”, por exemplo), mas quando usados para identificar um aplicativo, geralmente se utiliza de um nome de domínio “invertido” para evitar aplicativos diferentes usando o mesmo nome de pacote.



Diretiva *package*

- Um projeto pode ter vários pacotes, alguns deles compartilhando parte da estrutura hierárquica. Por exemplos, vamos considerar um projeto que tenha suas classes distribuídas entre os pacotes “br.edu.ifs.model” e “br.edu.ifs.dao”. Sua estrutura em pastas, seria a seguinte:



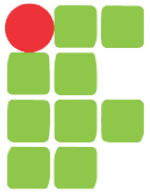
Diretiva *import*

- Especifica nome de pacotes e classes utilizadas naquele arquivo;
  - Pode referenciar pacotes e classes da própria plataforma Java bem como aqueles criados pelo próprio usuário;
- O objetivo aqui é informar ao compilador onde estão as classes necessárias para o funcionamento correto de nosso programa.

Diretiva *import*

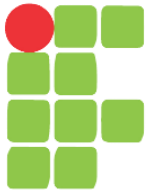
- Também aparecem no início do código de um arquivo (classe), uma linha para cada classe ou pacote de classes que desejamos importar. Exemplos:

```
import java.util.Scanner;
import java.util.*;
import br.edu.ifs.Sistema;
import br.edu.ifs.*;
```



Comentários

- Comentários são trechos de nosso código que serão ignorados pelo compilador, utilizados geralmente para:
  - “Ocultar” temporariamente parte do código;
  - Incluir explicações sobre o mesmo;
  - Incluir anotações/lembretes para realizar alguma ação futura.



Comentários

- Em Java, há dois tipos de comentários:

```
//Comentário em linha

/*Comentário em bloco
Para uma ou mais linhas!*/
```

Comentários

- Java reconhece as seguintes “task tags” (tags em comentários para apontar coisas específicas a serem feitas ali):

//TODO Este comentário indica algo a ser feito

//FIXME Este outro aponta algum problema no código

//XXX Este indica que o código funciona mas precisa ser revisado ou reescrito

- É possível definir outras task tags, por meio do menu Windows > Preferences..., na opção Java > Compiler > Task Tags.
- Tais comentários irão aparecer na aba/view Tasks.

Blocos de código

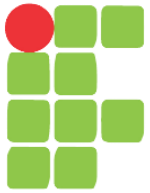
- Um bloco de código é utilizado para especificar uma série de comandos/instruções que devem ser executadas a partir de um outro comando (de condição ou repetição), função ou classe.

Blocos de código

- Blocos de código são delimitados por chaves “{“ “}”.

Exemplo 1:

```
public static void main(String[] args) {
    int num = 5 + 2;
    System.out.println(num);
}
```



Blocos de código

Exemplo 2:

```
float nota1 = 6.0f, nota2 = 7.0f, nota3 = 8.0f;
float media = (nota1 + nota2 + nota3) / 3;
if (media >= 6.0) {
    System.out.println("Aprovado");
} else {
    System.out.println("Reprovado");
}
```

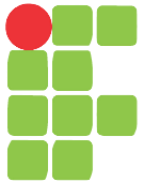
Palavras reservadas

- São palavras que possuem específicas dentro de uma linguagem de programação e, portanto, não podem ser usadas para outra finalidade (por exemplo, para declarar uma variável).

Palavras reservadas

- Alguns exemplos de palavras reservadas em Java:
  - Tipos de dados – int, float, char, boolean etc.
  - Comandos condicionais ou de repetição – if, else, switch, case, for, while etc.
  - Qualificadores – public, private, final, static etc.
  - E outras.

Exercícios



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Comandos para leitura e impressão

Parte 05

Sumário

- Comandos para leitura
- Comandos para impressão

Recordando...

- O que são pacotes?
- Para que serve a diretiva package?
- Para que serve a diretiva import?
- Como podemos escrever comentários em um código?

Comandos para leitura

- Há várias formas de um usuário entrar dados em um aplicativo (leitura de dados);
- Por agora, faremos a leitura a partir do console padrão do Eclipse, representado pelo objeto “in” presente na classe System;

System.in

- Para proceder com a leitura, Java oferece várias soluções (Scanner, BufferedReader, BufferedInputStream etc.). Aqui, utilizaremos a classe Scanner.

Comandos para leitura

1º passo: Crie (instancie) o objeto a ser utilizado na leitura dos dados

Scanner leitor = new Scanner(System.in);

2º passo: Faça a leitura de cada uma das informações para o seu programa, usando a função para o tipo correspondente

int idade = leitor.nextInt(); //Lê o próximo valor inteiro

float nota = leitor.nextFloat(); //Lê o próximo valor float

String nome = leitor.nextLine(); //Lê a próxima String

Comandos para impressão

- Para a escrita de dados na tela (impressão), utilizaremos o objeto “out” da classe System, que possui dois métodos para escrita bastante simples:

System.out.print(); //Escreve no console, mas não pula para a próxima linha

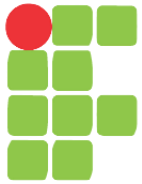
System.out.println(); //Escreve no console e pula para a próxima linha

Comandos para impressão

Exemplos:

```
System.out.print( “Primeira mensagem” );
System.out.println( “Segunda mensagem” );
System.out.println( “Terceira mensagem” );
```

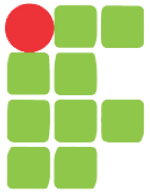
Exercícios



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Comandos condicionais

Parte 06



Sumário

- Comandos condicionais
- Comando if... else
- Comando switch
- Condicional “?”

Recordando...

- Quais as classes que usaremos para criar um objeto para leitura de dados (receber dados do usuário)?

- Quais os comandos disponíveis para escrever no console?

***Dica**: Sempre que for ler uma informação no console, escreva antes uma mensagem dizendo o que está sendo lido – fará mais sentido para o usuário.*

Comandos condicionais

- Em algumas situações, queremos que aconteçam coisas diferentes, para situações (condições) diferentes;

- Vejamos alguns exemplos...

Comandos condicionais

- Primeiro exemplo, um programa que imprime uma mensagem se um número é divisível por 2:
  - Leia um número N;
  - **SE** o número N é divisível por 2, **ENTÃO** escreva “<N> é divisível por 2”.

Comandos condicionais

- Outro exemplo, um programa que informa se um número é divisível por 2, por 3 ou por 5:
  - Leia um número N;
  - **SE** o número é divisível por 2, **ENTÃO** escreva “<N> é divisível por 2”;
  - **SE** o número é divisível por 3, **ENTÃO** escreva “<N> é divisível por 3”;
  - **SE** o número é divisível por 5, **ENTÃO** escreva “<N> é divisível por 5”.

Comandos condicionais

- Segundo exemplo, um programa que informa a situação de um aluno (aprovado ou reprovado, de acordo com a média):
  - Leia as três notas de um aluno;
  - Calcule a sua média;
  - **SE** a média do aluno for superior à média mínima (6.0), **ENTÃO** ele está aprovado;
  - **SENÃO**, ele está reprovado.

Comandos condicionais

- Outro exemplo, um programa que informa se um número é par ou não:
  - Leia um número N;
  - **SE** o número é divisível por 2, então é par;
  - **SENÃO**, ele é ímpar.

Comandos condicionais

- Para tais situações, existem os comandos condicionais, que são comandos (instruções) que permitem escolher executar ou não um bloco de código de acordo com uma condição;

- Cada linguagem implementa seus próprios comandos condicionais, Java apresenta três mais comuns:
  - O comando *if... else*
  - O comando *switch*
  - O comando *“?”*

Comando if...

- Comando presente em praticamente todas as linguagens de programação, ele permite testar se uma condição é verdadeira e, se sim, executa-se um bloco associado a ele.

Comando if...

Sintaxe:

```
if (condição) {
    comandos;
}
```

Leia-se:

```
SE (condição for verdadeira) ENTÃO {
    comandos;
}
```

Comando if...

Exemplos:

```
if (idade >= 18) {
    System.out.println("Você é maior de idade");
}

if (x == 5) {
    System.out.println("x é cinco!");
}

if (nome.equals("Pedro")) {
    System.out.println("Seja bem-vindo, Pedro!");
}
```

Comando if... – Exercícios

- Retorne aos slides 104 e 105 e tente implementar em Java algoritmos que satisfaçam o que se pede.

Comando if... else

- Em algumas situações, podemos querer executar outro bloco de código, caso a condição falhe. Nessas situações, devemos utilizar a palavra reservada **else** (“senão”) para expressar o que deve ser feito caso a condição do if seja falsa.

Comando if... else

Sintaxe:

```
if (condição) {
    comandos;
} else {
    comandos;
}
```

Leia-se:

```
SE (condição for verdadeira) ENTÃO {
    comandos;
} SENÃO {
    comandos;
}
```

Comando if... else

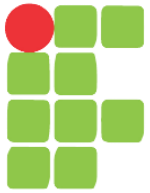
Exemplo 1:

```
if (idade >= 18) {
    System.out.println("Você é maior de idade");
} else {
    System.out.println("Você é menor de idade");
}
```

Comando if... else

Exemplo 2:

```
if (x == 5) {
    System.out.println("x é cinco!");
} else {
    System.out.println("x não é cinco!");
}
```



Comando if... else

Exemplo 3:

```
if (nome.equals("Pedro")) {
    System.out.println("Seja bem-vindo, Pedro!");
} else {
    System.out.println("Seja bem-vindo, senhor!");
}
```

Comando if... else - Exercícios

- Retorne aos slides 106 e 107 e tente implementar em Java algoritmos que satisfaçam o que se pede.

Comando if... else - Observações

- Podemos ter um if sem else, mas não podemos ter um else sem if!

```
if (x >=18) {
    System.out.println("Você é maior de idade");
} else {
    System.out.println("Você é menor de idade");
} else {
    System.out.println("Você não pode entrar!");
}
```

Comando if... else - Observações

- Podemos ter ifs aninhados (um if dentro do outro).

```
if (x >=18) {
    if (sexo == 'F') {
        System.out.println("Você já é uma mulher!");
    } else {
        System.out.println("Você já é um homem!");
    }
}
```

Comando if... else - Observações

- Podemos ter ifs encadeados (um dentro do senão do outro).

```
if (nota >= 10) {
    System.out.println("Sua nota foi perfeita!");
} else if (nota >= 9) {
    System.out.println("Sua nota foi muito boa!");
} else if (nota >= 7.5) {
    System.out.println("Sua nota foi boa!");
} else {
    System.out.println("É bom revisar um pouco mais.");
}
```

Comando switch

- Às vezes, queremos executar blocos de código de acordo com um valor escolhido ou calculado. Exemplo em um sistema:

```
Olá, usuário! O que você deseja?
1. Cadastrar novo produto
2. Listar produtos cadastrados
3. Cadastrar novo cliente
4. Listar clientes
5. Sair

Digite sua opção:
```

Comando switch

- No exemplo anterior, o código a ser executado será diferente, a depender da opção (valor numérico) escolhida pelo usuário;
- Nesta situação, dizemos que a condição vai depender de cada caso, isto é, de cada valor;
- Podemos implementar isso usando vários “if”, mas um jeito mais interessante é utilizando o comando condicional switch.

Comando switch

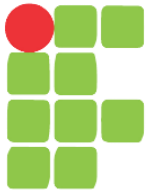
- O comando condicional switch permite que, a partir do valor de uma expressão (pode ser uma variável), seja executado o bloco de código associado àquele caso (valor/rótulo);
- Sua sintaxe é:

```
switch (expressao) {
    case valor1:
            comandos;
            break;
    case valor2:
            comandos;
            break;
    case valorN:
            comandos;
            break;
    default:
            comandos;
}
```

Comando switch

- Onde:
  - **switch** – palavra reservada que informa a expressão a ser calculada e, em seguida, os casos (rótulos) para o sistema testar;
  - **expressao** – aqui vai a expressão a ser avaliada (isto é, calculada). Seu resultado deve ser um número inteiro, String ou tipo enumerado;
  - **case** – informa o valor que, caso coincida com o resultado da expressão, os comandos presentes daquele ponto até o final do switch serão executados;

```
switch (expressao) {
    case valor1:
            comandos;
            break;
    case valor2:
            comandos;
            break;
    case valorN:
            comandos;
            break;
    default:
            comandos;
}
```



Comando switch

- Onde: (cont.)
  - **break** – realiza uma “quebra” na execução do switch, “pulando” para logo após o mesmo. Geralmente aparece após a lista de comandos de um *case*;
  - **default** – especifica uma série de comandos a serem executados, caso o resultado da expressão não “case” com nenhum dos valores passados. É opcional.

```
switch (expressao) {
    case valor1:
            comandos;
            break;
    case valor2:
            comandos;
            break;
    case valorN:
            comandos;
            break;
    default:
            comandos;
}
```

Comando switch - Exemplo

```
Scanner leitor = new Scanner(System.in);
System.out.println("Escolha o tipo de figura para calcular a área:");
System.out.println("1. Triângulo");
System.out.println("2. Quadrado");
System.out.println("3. Círculo");
int opcao = leitor.nextInt();

switch (opcao) {
    case 1:
        System.out.println("Informe o valor da base:");
        float base = leitor.nextFloat();
        System.out.println("Informe o valor da altura:");
        float altura = leitor.nextFloat();
        float area = base * altura / 2;
        System.out.println("A área do triângulo é: " + area);
        break;
    case 2:
        ...
    case 3:
        ...
}
```

Comando switch - Exercício

Crie um programa que permita ao usuário informar uma das seguintes conversões:

1. KB para MB;
2. KB para GB;
3. MB para KB;
4. GB para KB.

Em seguida, o usuário informará o valor na medida original e o sistema informará o valor correspondente na medida alvo.

Operador Condicional

- Há situações em que, caso uma condição seja verdadeira, queremos retornar um valor, caso contrário, outro;
- Podemos alcançar esse objetivo por meio da instrução if...else, mas também podemos fazê-lo por meio do operador condicional “?”, também conhecido como operador ternário.

Operador Condicional

Sintaxe:

condição ? valor_se_verdadeiro : valor_se_falso

Exemplo:

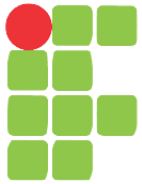
int a = (b > 0 ? b : 0);

Operador Condicional

Exemplo: Ler um número e imprimir o dobro se ele for positivo, a metade se ele for negativo.

Código Java:	**Leia-se:**
float n = leitor.nextFloat(); *float resposta = (n > 0 ? n*2 : n/2);* *System.out.println(resposta);*	*LEIA n;* *SE n > 0 ENTÃO resposta = n*2;* *SENÃO resposta = n/2;* *ESCREVA resposta;*

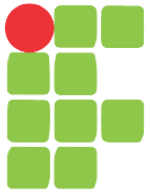
Exercícios



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SERGIPE

Comandos de repetição

Parte 07



Sumário

- Comandos de repetição
- Comando for
- Comando while
- Comando do... while

Recordando...

- Para que servem os comandos condicionais?
- Qual a diferença entre usar um if e um if...else?
- Para que server a instrução switch?

Comandos de repetição

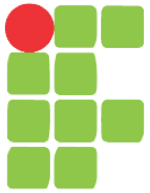
- Para resolver alguns problemas, precisamos repetir um conjunto de instruções.
  Exemplos:
  - Imprimir todos os números de 1 a 100;
  - Imprimir somente os números pares de 1 a 100;
  - Imprimir a soma de todos os números de 1 a 100;
  - Receber dois números inteiros e imprimir todos os inteiros do maior até o menor deles.

Comandos de repetição

- Para cada uma dessas situações, queremos executar uma série de passos (instruções) para cada número daquele intervalo, a fim de encontrarmos a solução;
- Assim, precisamos conhecer comandos de repetição, instruções que permitem repetir um bloco de código sob determinadas circunstâncias;
- Java oferece três instruções para repetição: for, while e do...while.

Comando for

- Este comando de repetição permite especificar uma variável de controle que será usada para “varrer” os números de um intervalo (de 5 até 10, por exemplo) e como essa variável deve ser incrementada/decrementada ao final de cada passo.



Comando for

Sintaxe:

```
for (inicialização; condição; incremento) {
    comandos;
}
```

Comando for

Exemplo 1: Imprimir todos os números de 1 a 100

Código Java:	**Leia-se:**
for (int i = 1; i <= 100; i++) { *System.out.println(i);* *}*	*PARA int i = 1; ENQUANTO i <= 100 FAÇA {* *ESCREVA i;* *} (incremente i em 1);*

Comando for

Exemplo 2.a: Imprimir todos os números pares de 1 a 100

Código Java:

```
for (int i = 1; i <= 100; i++) {
   if (i % 2 == 0) {
      System.out.println(i);
   }
}
```

Leia-se:

```
PARA int i = 1; ENQUANTO i <= 100 FAÇA {
   SE (resto de 1 dividido por 2  == 0) ENTÃO {
      ESCREVA  i;
   }
} (incremente   i em 1);
```

Comando for

Exemplo 2.b: Imprimir todos os números pares de 1 a 100

Código Java:	**Leia-se:**
for (int i = 2; i <= 100; i+=2) { *System.out.println(i);* *}*	*PARA int i = 2; ENQUANTO i <= 100 FAÇA {* *ESCREVA i;* *} (incremente i em 2);*

Comando for

Exemplo 3: Imprimir soma de todos os números de 1 a 100

Código Java:	**Leia-se:**
int soma = 0; *for (int i = 1; i <= 100; i++) {* *soma = soma + i;* *}* *System.out.println(soma);*	*int soma = 0;* *PARA int i = 1; ENQUANTO i <= 100 FAÇA {* *soma = soma + i;* *} (incremente i em 1);* *ESCREVA soma;*

Comando for

Exemplo 4: Receber dois números inteiros e imprimir todos os inteiros do maior até o menor deles

Código Java:

```
int a = leitor.nextInt();
int b = leitor.nextInt();
int maior = (a > b ? a : b);
int menor = (a < b ? a : b);
for (int x = maior; x >= menor; x--)
{
   System.out.println(x);
}
```

Leia-se:

```
LEIA a;
LEIA b;
int maior = maior valor entre a e b;
int menor = menor valor entre a e b;
PARA int x = maior; ENQUANTO x >= menor FAÇA
{
   ESCREVA x;
} (decremente   x em 1);
```

Comando for - Exercícios

- Escreva um programa que recebe dois números inteiros e imprime na tela a soma de todos os números daquele intervalo;
- Escreva um programa que recebe um número inteiro e imprime na tela o seu fatorial, sendo que:
  - Fatorial de números negativos não existe
  - Fatorial de 0 = 1
  - Fatorial de n = 1*2*3*...*n
- Escreva um programa que recebe 10 números e, ao final, imprime na tela o maior deles.

Comando while

- Este comando de repetição permite especificar uma condição de controle e, enquanto for verdadeira, repetirá o bloco de código vinculado.

Comando while

Sintaxe:

```
while (condição) {
    comandos;
}
```

Comando while

Exemplo 1: Imprimir todos os números de 1 a 100

Código Java:	**Leia-se:**
int i = 1; *while (i <= 100) {* *System.out.println(i);* *i++;* *}*	*int i = 1;* *ENQUANTO i <= 100 FAÇA {* *ESCREVA i;* *INCREMENTE i em 1;* *}*

Comando while

Exemplo 2.a: Imprimir todos os números pares de 1 a 100

Código Java:

```
int i = 1;
while (i <= 100) {
   if (i % 2 == 0) {
      System.out.println(i);
   }
   i++;
}
```

Leia-se:

```
int i = 1;
ENQUANTO i <= 100 FAÇA {
   SE (resto de 1 dividido por 2  == 0) ENTÃO {
      ESCREVA  i;
   }
   INCREMENTE  i em 1;
}
```

Comando while

Exemplo 2.b: Imprimir todos os números pares de 1 a 100

Código Java:

```
int i = 2;
while (i <= 100) {
   System.out.println(i);
   i += 2;
}
```

Leia-se:

```
int i = 2;
ENQUANTO i <= 100 FAÇA {
   ESCREVA  i;
   INCREMENTE i em 2;
}
```

Comando while

Exemplo 3: Imprimir soma de todos os números de 1 a 100

Código Java:

```
int soma = 0;
int i = 1;
while (i <= 100) {
   soma = soma + i;
   i++;
}
System.out.println(soma);
```

Leia-se:

```
int soma = 0;
int i = 1;
ENQUANTO i <= 100 FAÇA {
   soma = soma + i;
   INCREMENTE i em 1;
}
ESCREVA soma;
```

Comando while

Exemplo 4: Receber dois números inteiros e imprimir todos os inteiros do maior até o menor deles

Código Java:

```
int a = leitor.nextInt();
int b = leitor.nextInt();
int maior = (a > b ? a : b);
int menor = (a < b ? a : b);
int x = maior;
while (x >= menor) {
   System.out.println(x);
   x--;
}
```

Leia-se:

```
LEIA a;
LEIA b;
int maior = maior valor entre a e b;
int menor = menor valor entre a e b;
int x = maior;
ENQUANTO x >= menor FAÇA {
   ESCREVA x;
   DECREMENTE x em 1;
}
```

Comando while - Exercícios

- Escreva um programa que recebe um número inteiro e imprime na tela o seu fatorial, sendo que:
  - Fatorial de números negativos não existe
  - Fatorial de 0 = 1
  - Fatorial de n = 1*2*3*...*n
- Escreva um programa que calcule a soma de todos os números positivos recebidos. Quando o usuário informar um número negativo ou zero, imprima a soma encontrada e encerre o programa.

Comando do...while

- Em certas situações, queremos que o bloco de código a ser repetido seja executado antes de testar a condição de controle, obrigando assim aquele bloco a ser executado pelo menos uma vez;
- Nessas circunstâncias, podemos utilizar a instrução *do...while*, que trabalha de forma similar ao *while*, porém testando a condição de controle somente após executar o bloco.

Comando do...while

Sintaxe:

```
do {
    comandos;
} while (condição);
```

Comando do...while

Exemplo 1: Imprimir todos os números de 1 a 100

Código Java:	**Leia-se:**
int i = 1;	*int i = 1;*
do {	*FAÇA {*
System.out.println(i);	*ESCREVA i;*
i++;	*INCREMENTE i em 1;*
} while (i <= 100);	*} ENQUANTO i <= 100;*

Comando do...while

Exemplo 2.a: Imprimir todos os números pares de 1 a 100

Código Java:

```
int i = 1;
do {
   if (i % 2 == 0) {
      System.out.println(i);
   }
   i++;
} while (i <= 100);
```

Leia-se:

```
int i = 1;
FAÇA {
   SE (resto de i dividido por 2  == 0) ENTÃO {
      ESCREVA  i;
   }
   INCREMENTE  i em 1;
} ENQUANTO i <= 100;
```

Comando do...while

Exemplo 2.b: Imprimir todos os números pares de 1 a 100

Código Java:	**Leia-se:**
int i = 2; *do {* *System.out.println(i);* *i += 2;* *} while (i <= 100);*	*int i = 2;* *FAÇA {* *ESCREVA i;* *INCREMENTE i em 2;* *} ENQUANTO i <= 100;*

Comando do...while

Exemplo 3: Imprimir soma de todos os números de 1 a 100

Código Java:	**Leia-se:**
int soma = 0;	*int soma = 0;*
int i = 1;	*int i = 1;*
do {	*FAÇA {*
soma = soma + i;	*soma = soma + i;*
i++;	*INCREMENTE i em 1;*
} while (i <= 100);	*} ENQUANTO i <= 100;*
System.out.println(soma);	*ESCREVA soma;*

Comando do...while

Exemplo 4: Receber dois números inteiros e imprimir todos os inteiros do maior até o menor deles

Código Java:

```
int a = leitor.nextInt();
int b = leitor.nextInt();
int maior = (a > b ? a : b);
int menor = (a < b ? a : b);
int x = maior;
do {
   System.out.println(x);
   x--;
} while (x >= menor);
```

Leia-se:

```
LEIA a;
LEIA b;
int maior = maior valor entre a e b;
int menor = menor valor entre a e b;
int x = maior;
FAÇA {
   ESCREVA x;
   DECREMENTE x em 1;
} ENQUANTO x >= menor;
```

Comparando *while* e *do...while*

Nos casos abaixo, o que acontecerá se o valor passado para “a” for...
5? 1? 0? -5?

Código usando *while*:

```
int a = leitor.nextInt();
while (a > 0) {
   System.out.println(a);
   a--;
}
```

Código usando *do...while*:

```
int a = leitor.nextInt();
do {
   System.out.println(a);
   a--;
} while (a > 0);
```

Comando do...while - Exercícios

- Escreva um programa que recebe um número inteiro e imprime na tela o seu fatorial, sendo que:
  - Fatorial de números negativos não existe
  - Fatorial de 0 = 1
  - Fatorial de n = 1*2*3*...*n
- Escreva um programa que calcule a soma de todos os números positivos recebidos. Quando o usuário informar um número negativo ou zero, imprima a soma encontrada e encerre o programa.

Comando do...while - Exercícios

- Crie um programa que permita ao usuário informar uma das seguintes opções:

1. KB para MB;
2. KB para GB;
3. MB para KB;
4. GB para KB;
0. Sair.

Caso usuário escolha uma opção de 1 a 4, ele informará o valor na medida original e o sistema informará o valor correspondente na medida alvo.

Caso informe uma opção inválida (5, por exemplo), o sistema emitirá mensagem dizendo que a opção escolhida é inválida. Em ambos os casos, após processar a opção escolhida, o programa imprimirá novamente as opções e o usuário poderá informar nova opção.

O programa somente encerrará quando o usuário informar a opção “0” (sair).

Comandos para interrupção de laços (repetições)

- Em algumas situações, podemos querer interromper um comando de repetição antes que ele alcance a condição de parada, por exemplo quando encontra um valor esperado;
- Nessas situações, temos duas opções de instrução para interrupção do laço: **break** e **continue**.

Comando break

- Quando executado, interrompe a execução dos comandos dentro do laço (*for*, *while* ou *do...while*) ou *switch* e sai imediatamente do mesmo.

Comando break

Exemplo:

```
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0) {
        break;
    }
    System.out.println(i + " não é divisível por 3");
}
System.out.println("===Programa encerrado===");
```

Saída:

```
1 não é divisível por 3
2 não é divisível por 3
===Programa encerrado===
```

Comando continue

- Quando executado, interrompe a execução dos comandos dentro do laço (*for*, *while* ou *do...while*), indo para o final do mesmo, executando assim a seção de incremento/decremento (caso o comando seja um for) e testando novamente a condição para continuar repetindo.

Comando continue

Exemplo:

```
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0) {
        continue;
    }
    System.out.println(i + " não é divisível por 3");
}
System.out.println("===Programa encerrado===");
```

Saída:

```
1 não é divisível por 3
2 não é divisível por 3
4 não é divisível por 3
5 não é divisível por 3
7 não é divisível por 3
8 não é divisível por 3
10 não é divisível por 3
===Programa encerrado===
```

Quando usar break ou continue?

- Em comandos switch, use *break*;
- Em comandos de repetição, use *break* se, naquele ponto, desejar interromper completamente a execução do laço e seguir com o programa...
- ... ou use *continue*, caso queira interromper somente a execução atual, “pulando” para a próxima.