Lógica de Programação

Linguagem de Programação Java

Prof. Luiz Alberto Parra

Linguagem de Programação
Questões

- O que é Linguagem de Programação?
- Defina Programa Fonte
- O que é Programa Objeto ou Executável?
- O que faz um Compilador?
- Relacionar três Linguagens de Programação e suas respectivas aplicações

Estrutura de um Programa

- Do ponto de vista da descrição, isto é, da forma utilizada para indicar ao computador o que ele deve fazer, um programa é composto, em geral, de duas partes:
  - Descrição de Dados
  - Descrição do Algoritmo

Estrutura de um Programa

- **Descrição de Dados:**
  - Local onde são declarados os dados utilizados no programa, tais como: variáveis, constantes, arquivos, tabelas, etc., indicando o nome, tipo, tamanho e demais características que o processador precisa conhecer.
- **Descrição do Algoritmo:**
  - Conjunto de instruções ou ações que o processador deverá realizar, na sequência lógica estabelecida.

Programa - Como Resolver Planejamento Reverso

1. Resultado: Variáveis de Saída.
2. Como resolver:
    1. Estruturas.
    2. Operações / Expressões.
3. Quem vai participar da solução:
    1. Variáveis de Entrada.
    2. Variáveis Auxiliares (contadores, acumuladores).
    3. Variáveis de Saída.

Fases para a Confecção de um Programa

1. **ANÁLISE DO PROBLEMA**

   Estudar minuciosamente o problema, suas entradas, saídas e procedimentos.

2. **ESQUEMATIZAR A SOLUÇÃO**

   Utiilizar pseudo-linguagem, diagrama de blocos, tabela de decisão, árvore de decisão.

3. **TESTAR A LÓGICA - TESTE DE MESA**

   Teste de validade das condições e procedimentos no esquema da solução.

Fases para a Confecção de um Programa

4. **CODIFICAÇÃO DO PROGRAMA**

   Escrever o programa em linguagem de programação.

5. **COMPILAÇÃO**

   Testar no computador a validade da linguagem.

6. **TESTE DO PROGRAMA**

   Carregar o programa na memória do computador, fornecer dados de Entrada e verificar os resultados.

7. **DOCUMENTAÇÃO**

   Relatar todos os procedimentos do programa.

Linguagem de Programação

- Uma linguagem qualquer é um conjunto de palavras e regras de sintaxe que devem ser obedecidas para exprimir uma dada ação.

- Em processamento de dados, Linguagem de Programação é um conjunto de convenções e regras que especificam como transmitir informações entre pessoas e máquinas, especificam como instruir o computador e executar determinadas tarefas.

.

Linguagem de Programação

- A Linguagem de Programação deve fazer a ligação entre o pensamento humano (nem sempre estruturado) e a precisão requerida para o processamento pela máquina/computador.

- O desenvolvimento de um programa será mais fácil se a Linguagem de Programação a ser usada estiver mais próxima do problema a ser resolvido.

Linguagem de Programação

- **Programa Fonte / Código Fonte:**
  - **É o programa escrito na linguagem de programação original e que precisa ser traduzido em linguagem de máquina para poder ser executado.**

Linguagem de Programação

- **Programa Objeto / Código Objeto / Programa Executável:**
  - **É o resultado da tradução de um programa fonte, quando usamos um COMPILADOR.**
  - **Antes de executar as instruções do programa fonte, o compilador traduz todas as instruções e gera um novo programa em LINGUAGEM DE MÁQUINA para, então executá-lo.**

Linguagem de Programação

- **Programa Objeto / Código Objeto / Programa Executável:**
  - **O processo de tradução pode ser realizado por dois tipos de tradutores: COMPILADORES e INTERPRETADORES.**
  - **Ambos tem a mesma função, diferindo no modo de realizar o processo de tradução.**

Linguagem de Programação

- **Compilador:**
  - **Lê, analisa e traduz todo o Programa Fonte, criando o Programa Objeto / Programa Executável, que corresponde às instruções em linguagem de máquina, necessárias para se atingir o objetivo do programa.**

Linguagem de Programação

- **Interpretador:**
  - Traduz cada comando de um programa em tempo de execução.
  - Utilizados para linguagens interativas ou conversacionais.
  - As vantagens do Interpretador estão relacionadas com a facilidade no desenvolvimento dos programas, mas eles precisam ser traduzidos toda vez que forem executados, uma vez que executam direto do programa fonte.
  - O inverso ocorre com os Compiladores que permitem, após a tradução, armazenar o Programa Objeto em arquivo apropriado, agilizando o processo de execução.

Linguagem JAVA

- É uma linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems.
- Modelada depois de C++, a linguagem Java foi desenvolvida para eletrodomésticos, portanto, projetada para ser pequena, simples e portável a todas as plataformas e sistemas operacionais, tanto o código fonte como os binários.
- Esta portabilidade é obtida pelo fato da linguagem ser interpretada, ou seja, o compilador gera um código independente de máquina, chamado *byte-code*.

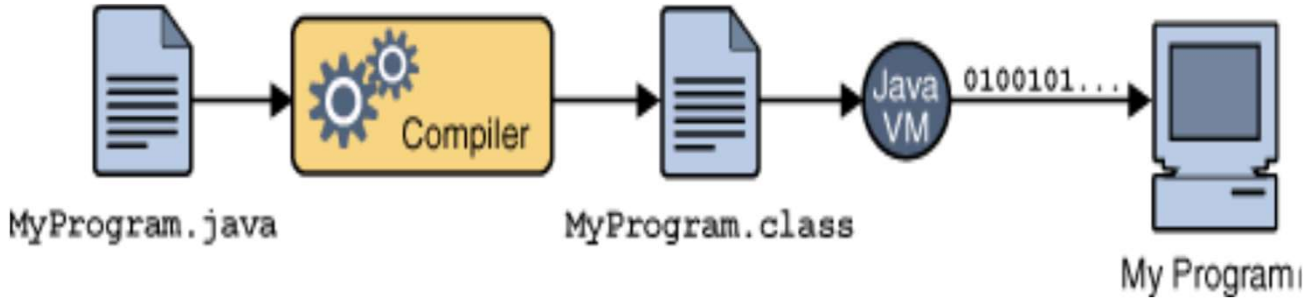
Linguagem JAVA

- No momento da execução, este *byte-code* é interpretado por uma máquina virtual instalada na máquina.
- Para portar Java para uma arquitetura de hardware específica, basta instalar a máquina virtual (interpretador).
- Além de ser integrada à Internet, Java também é uma excelente linguagem para desenvolvimento de aplicações em geral.
- Dá suporte ao desenvolvimento de software em larga escala.

FUNCIONAMENTO DA LINGUAGEM JAVA

- Compilação do Fonte (.java) para *byte-code* da JVM - Java Virtual Machine.
- Interpretação e execução do *byte-code* (.class).
- “Escreva uma vez, execute em qualquer lugar”.

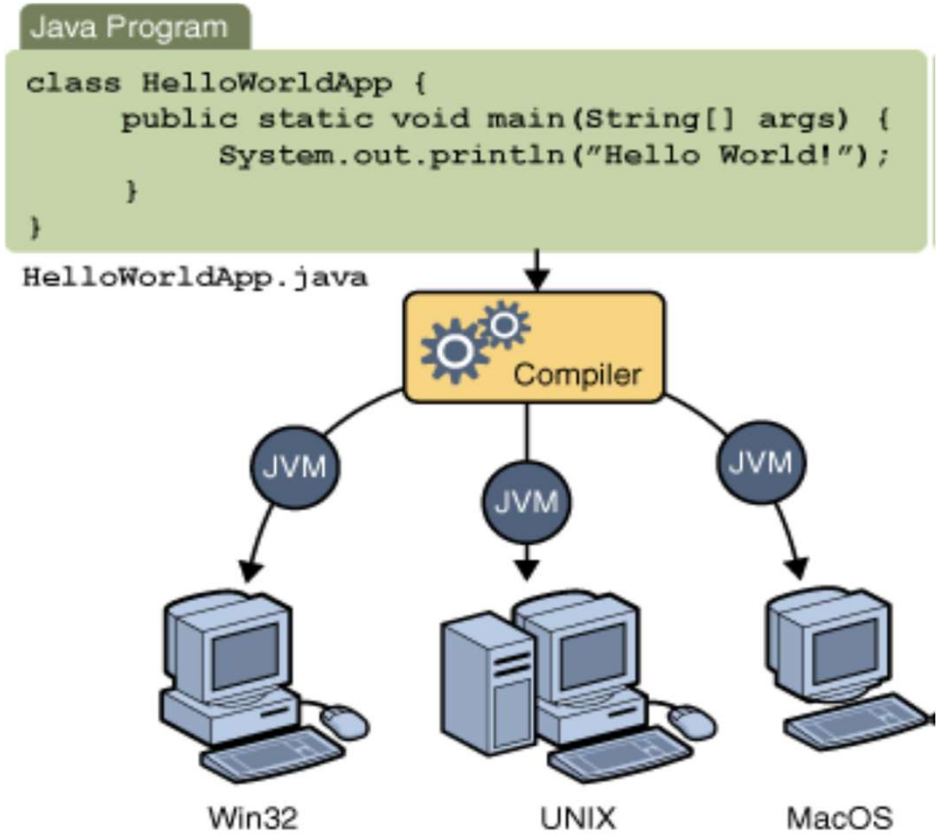
FUNCIONAMENTO DA LINGUAGEM JAVA



OUTRAS VANTAGENS DA LINGUAGEM JAVA

- Facilidades para desenvolvimento de aplicações em redes com protocolo TCP/IP (sockets, datagrama).
- Gerência automática de memória (*garbage collection*).
- Vários fornecedores de ambientes de desenvolvimento.
- Portabilidade: Independência de plataforma de hardware e software.
- Escalabilidade: Se for necessário colocar o sistema construído numa máquina mais robusta, provavelmente terá Java naquela máquina.

OUTRAS VANTAGENS DA LINGUAGEM JAVA



PROGRAMAS DE CRIAÇÃO DE PROGRAMAS

- Cada linguagem de programação poderá utilizar um programa ou ambiente para seu desenvolvimento.
- Exemplos:
  - Java: Netbeans, Eclipse, JDK, Blue J, Sun J2SE, Eclipse, Sun ONE Studio, JBuilder, VisualAge, Dr. Java , EJBWizard, CCM Core, JCreator Light Edition;
  - Pascal: Turbo Pascal.
  - Portugol: VisualG
  - C/C++: Code Blocks, Dev C++, Microsoft Visual C/C++ Express

PROGRAMAS DE CRIAÇÃO DE PROGRAMAS

- Para se criar um programa executável é necessário seguir alguns passos:
  - Redigir o programa através de um editor de textos, gravar em disco e nomear o arquivo - Código Fonte.
  - Compilar o Código Fonte, identificando e corrigindo os erros.
  - Se necessário, link editar para gerar o executável do seu programa.
- Observação: Para compilar e linkeditar uma única vez poderá utilizar o Netbeans IDE - Integrated Development Environment ou Eclipse.

ETAPAS DO DESENVOLVIMENTO DE UMA APLICAÇÃO JAVA

- As etapas que compõem o processo de desenvolvimento de uma aplicação na linguagem Java, desde o ambiente de desenvolvimento de aplicações da linguagem, até a execução do programa pelo usuário final, são:
  - Etapa 1 - Criação / Edição
  - Etapa 2 - Compilação
  - Etapa 3 - Carregamento
  - Etapa 4 - Verificação
  - Etapa 5 - Execução

ETAPA 1 - CRIAÇÃO / EDIÇÃO DE UM PROGRAMA JAVA

- Nesta primeira fase se dá a criação ou edição de arquivos em um Programa Editor, onde são inseridos os códigos pelo programador, e posteriormente salvos em uma unidade física de armazenamento como um programa de extensão “.java”.

Figura 1: O programa é criado no editor salvo com a terminação .java.

ETAPA 1 - CRIAÇÃO / EDIÇÃO DE UM PROGRAMA JAVA

- Para criarmos uma aplicação em Java, é necessário que utilizemos um programa de edição conhecido como Ambiente de Desenvolvimento Integrado, ou IDE (do inglês Integrated Development Enviroment).

- As IDE´s fornecem ferramentas que suportam o processo de desenvolvimento do software, incluindo editores para escrever e editar programas, e depuradores para localizar erros de lógica e sintaxe.

- Exemplos de IDE´s: Eclipse, NetBeans, Notepad ++ e outros.

ETAPA 2 - COMPILAÇÃO DO PROGRAMA JAVA EM BYTECODES

- Supondo que em um exemplo geramos um arquivo chamado “Ola.java”, na hora da compilação, o compilador produz um arquivo “.class” chamado “Ola.class” onde contém a versão compilada.
- O compilador Java converte o código-fonte em bytecodes que representam tarefas a serem executadas na fase de execução, melhor detalhada na Etapa 5.

Figura 2: O compilador cria bytecodes, armazenados com a extensão .class.

ETAPA 2 - COMPILAÇÃO DO PROGRAMA JAVA EM BYTECODES

- Os bytecodes são executados pela Java Virtual Machine - JVM, uma parte do JDK - Java Development Kit, ou Kit de Desenvolvimento Java, um conjunto de utilitários que permitem criar sistemas de software para a plataforma Java, composto por compilador e bibliotecas.

ETAPA 2 - COMPILAÇÃO DO PROGRAMA JAVA EM BYTECODES

- A Máquina Virtual, Virtual Machine – VM, é um aplicativo software que simula um computador, mas oculta o sistema operacional e o hardware subjacente dos programas que interagem com ela, o que garante a possibilidade de execução de programas Java em vários sistemas operacionais distintos.
- Características dos bytecodes:
  - Independentes de plataforma
  - Portáveis

ETAPA 3 - CARREGANDO UM PROGRAMA NA MEMÓRIA

- A JVM armazena o programa na memória para executá-lo, efetuando o carregamento.

- O carregador de classe da JVM pega os arquivos “.class”, que contêm os bytecodes do programa, e os transfere para a memória primária.

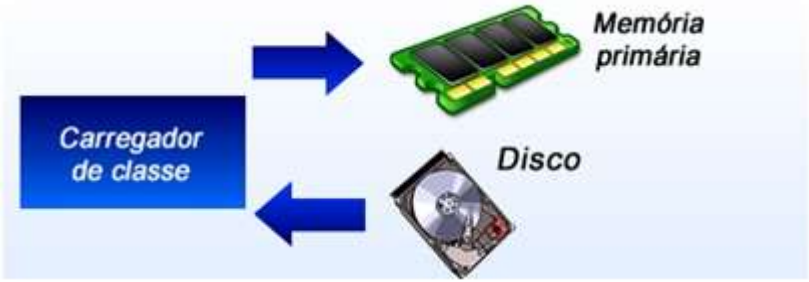


Figura 3: O carregador da classe lê os bytecodes e os aloca na memória

ETAPA 4 - VERIFICAÇÃO DO BYTECODE

- Enquanto as classes são carregadas, o Verificador examina seus bytecodes para assegurar que são válidos e não violam restrições de segurança do Java.

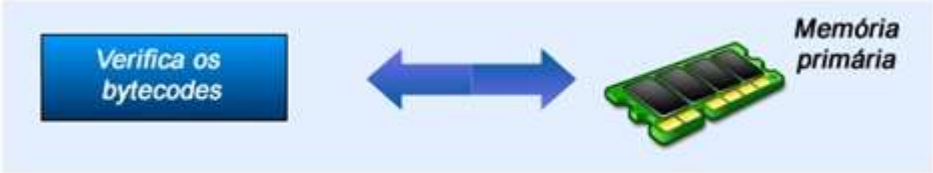


Figura 4: Verificação da integridade dos bytecodes

ETAPA 5 - EXECUÇÃO

- As JVMs executam os bytecodes, utilizando uma combinação de interpretação chamada compilação JIT (Just In Time) - conhecido como compilador Java HotSpot.

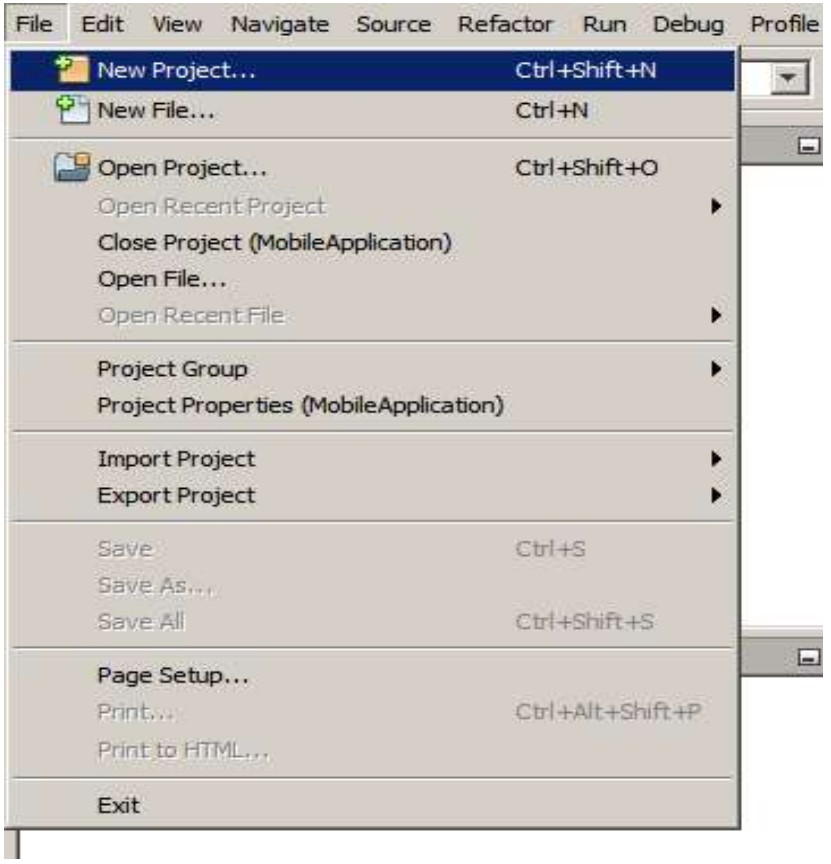
Figura 5: A JVM faz a leitura dos bytecodes e compila para uma linguagem de computador

ETAPA 5 - EXECUÇÃO

- O Java HotSpot traduz os bytecodes para a linguagem de máquina, e quando a JVM encontra novamente essas partes compiladas, o código de linguagem de máquina é executado mais rápido.
- Uma explicação mais detalhada é que o programa Java passa por duas fases de compilação que são:
  - Quando o código-fonte é traduzido em bytecodes, que acaba tendo uma portabilidade entre JVMs em diferentes plataformas do computador.
  - E a outra é durante a execução, quando os bytecodes são traduzidos em linguagem de máquina para o computador real em que o programa é executado.

JAVA - CONFIGURANDO O PROJETO

- Inicie o NetBeans IDE.
- Escolha Arquivo > Novo Projeto;



JAVA - CONFIGURANDO O PROJETO

- No assistente Novo Projeto, expanda a categoria Java e selecione Aplicação Java; Clique em Next - Próximo;

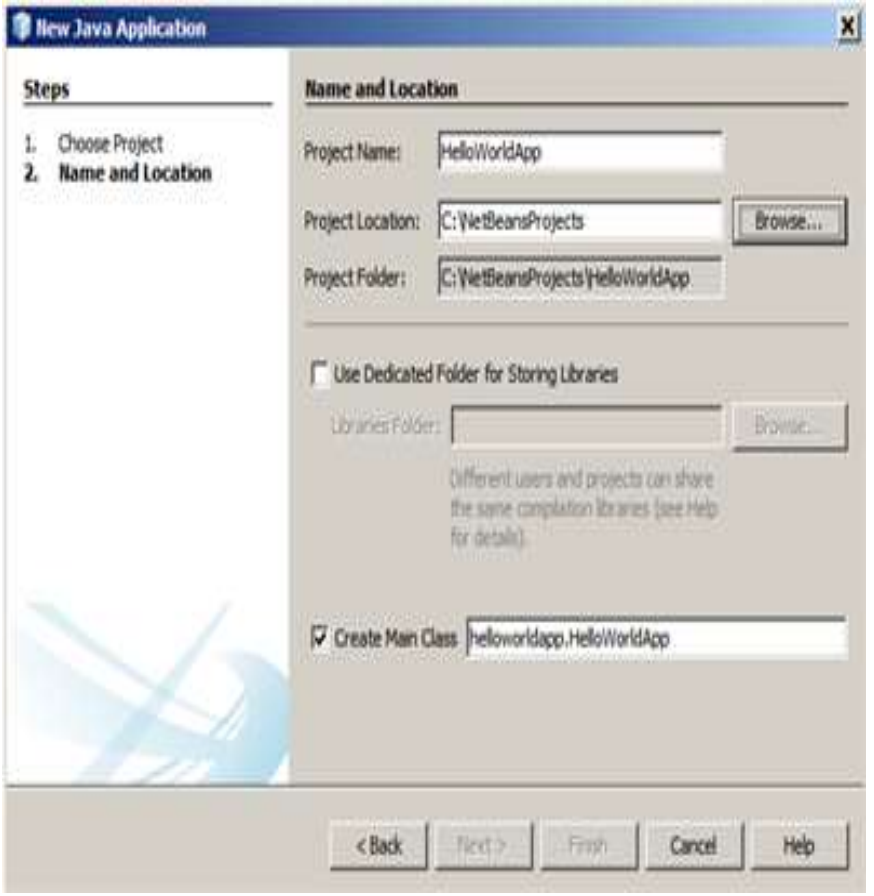
JAVA - CONFIGURANDO O PROJETO

- Na página Nome e Localização do assistente, no campo Nome do Projeto, digite HelloWorldApp;

- Deixe desmarcada a caixa de seleção Utilizar Pasta Dedicada para Armazenar Bibliotecas;

JAVA - CONFIGURANDO O PROJETO

- No campo Criar Classe Principal, digite helloworldapp.HelloWorldApp.
- Clique em Finalizar.



JAVA - CONFIGURANDO O PROJETO

Agora você pode ver:

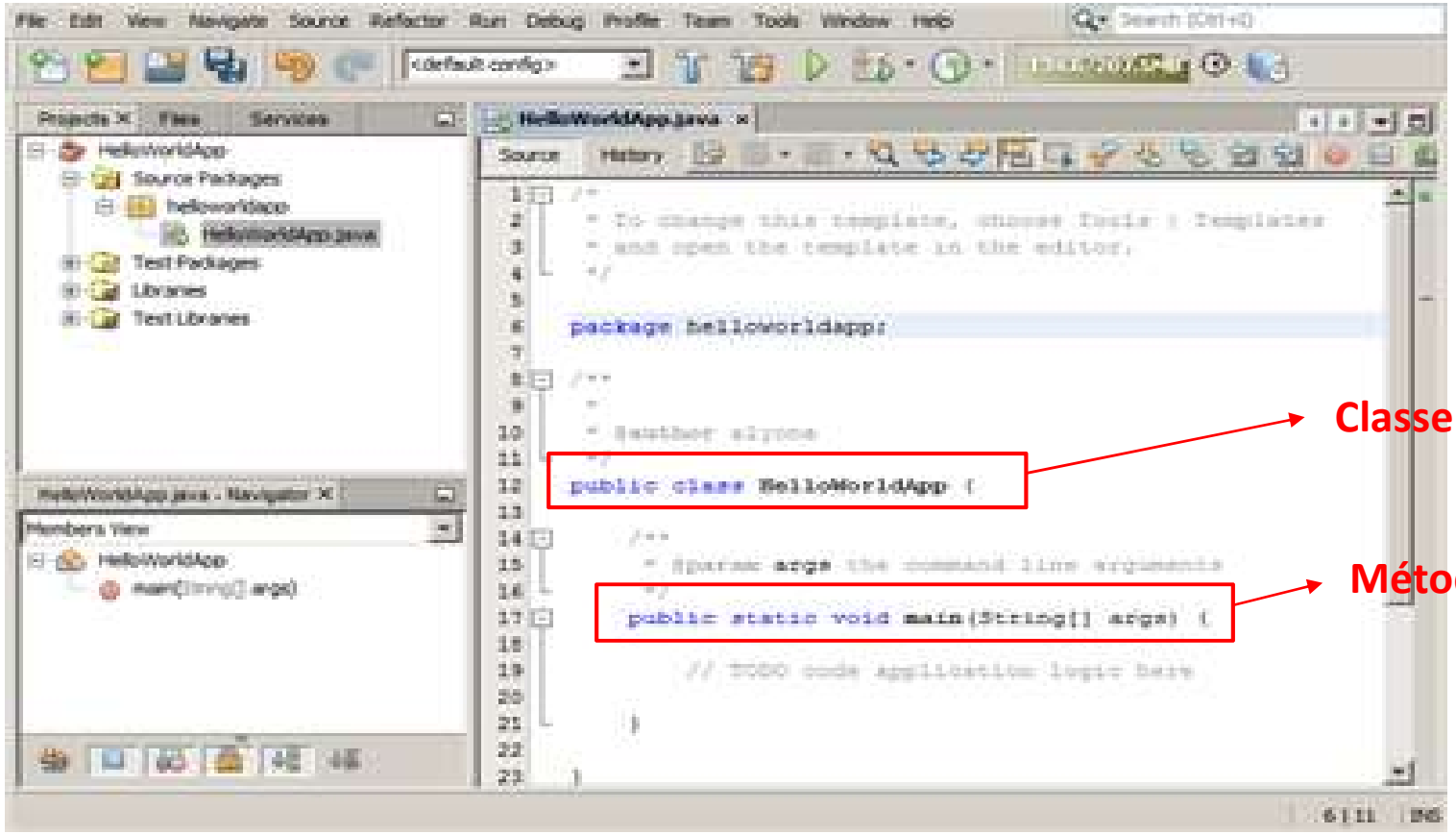
- A janela Projetos, que contém uma *view* em árvore dos componentes do projeto, incluindo arquivos de código-fonte, bibliotecas de que seu código depende, e assim por diante;

JAVA - CONFIGURANDO O PROJETO

Agora você pode ver:

- A janela Editor de Código-fonte com um arquivo chamado HelloWorldApp;
- A janela Navegador, por meio da qual se pode navegar rapidamente entre elementos dentro da classe selecionada.

JAVA - CONFIGURANDO O PROJETO



PROGRAMA EM JAVA
HELLO WORLD


```
package helloworldapp;

public class HelloWorldApp {

    public static void main(String[] args) {
        System.out.println("Hello World!!!");
    }

}
```

JAVA - Package

- Pacotes são definidos e utilizados para organizar as classes semelhantes;

- Pacotes são, a grosso modo, pastas ou diretórios do sistema operacional onde ficam armazenados os arquivos fonte de Java.

JAVA - Public Class

- Define um conjunto de objetos com características comuns;
- Uma classe é como um modelo para a criação de objetos que tem as mesmas características da classe à qual pertencem.

JAVA - Public Static Void Main

- Consiste no método principal de uma aplicação JAVA;
- Em uma aplicação JAVA podem existir muitas classes e muitos métodos;
- Quando se executa uma aplicação JAVA, o método **MAIN** é o ponto de entrada da aplicação.

ELEMENTOS BÁSICOS DA SINTAXE

COMANDOS

- Comandos em Java são separados por um ponto-vírgula (;).
- É permitido colocar vários comandos em uma mesma linha.
  Observação: O ideal é termos um comando por linha.
- Comandos são organizados em blocos definidos por chaves {.....}.

ELEMENTOS BÁSICOS DA SINTAXE

COMANDOS

- Em especial, classes e métodos são blocos - blocos podem conter outros blocos.
- Os espaços em branco *são* permitidos entre os elementos do código-fonte, sem qualquer restrição - espaços, tabulações e novas linhas, e são usados para melhorar a aparência visual e a legibilidade do código-fonte.
- Java é *case sensitive*, ou seja, maiúsculas são diferentes de minúsculas. Exemplo: TRUE é diferente de true.

ELEMENTOS BÁSICOS DA SINTAXE

VARIÁVEIS

- Variáveis são alocações de memória nas quais podemos guardar dados.
- Elas têm um Nome, Tipo/Formato e Valor.
- Toda vez que necessite usar uma variável, você precisa declará-la e só então poderá atribuir valores à mesma.
- O escopo de uma variável, ou seja a região do programa na qual ela está definida, é limitado ao bloco no qual ela foi declarada.

ELEMENTOS BÁSICOS DA SINTAXE

VARIÁVEIS

- As declarações de variáveis consistem de um Tipo/Formato e um Nome de variável: como segue o exemplo:
  - int idade;
  - string nome;
  - boolean existe;
- Os nomes de variáveis podem começar com uma letra, um sublinhado (_), ou um cifrão ($).
- Elas não podem começar com um número.

ELEMENTOS BÁSICOS DA SINTAXE

VARIÁVEIS

- Depois do primeiro caracter pode-se colocar qualquer letra ou número.
- É convencional (mas não obrigatório) usar uma letra minúscula para a primeira letra do nome de uma variável.
- Não pode utilizar palavras reservadas da programação Java.

ELEMENTOS BÁSICOS DA SINTAXE

TIPOS/FORMATOS DE VARIÁVEIS

- Toda variável deve possuir um Tipo/Formato.
- Os tipos que uma variável pode assumir uma das três "coisas" a seguir:
  - Um dos oito tipos primitivos básicos de dados
  - O nome de uma classe ou interface
  - Um array
- Veremos mais sobre o uso de arrays e classes mais à frente.

ELEMENTOS BÁSICOS DA SINTAXE
TIPOS/FORMATOS DE VARIÁVEIS

- Tipos Inteiros:

TIPO	TAMANHO	ALCANCE
byte	8 bits	- 128 até 127
short	16 bits	- 32.768 até 32.767
int	32 bits	- 2.147.483.648 até 2.147.483.647
long	64 bits	- 9.223.372.036.854.775.808 até 9.223.372.036.854.775.807

ELEMENTOS BÁSICOS DA SINTAXE
TIPOS/FORMATOS DE VARIÁVEIS

- **Tipos Reais - Ponto Flutuante:**

TIPO	TAMANHO	ALCANCE
float	32 bits	- 3.40292347E+38 até +3.40292347E+38 Aproximadamente 6 a 7 dígitos decimais PRECISÃO SIMPLES
double	64 bits	- 1.79769313486231570E+308 até +1.79769313486231570E+308 Aproximadamente 15 dígitos decimais PRECISÃO DUPLA

ELEMENTOS BÁSICOS DA SINTAXE
TIPOS/FORMATOS DE VARIÁVEIS

- **Tipo Caracter:**

TIPO	TAMANHO	ALCANCE
char	16 bits	‘\u0000’ até ‘\uFFFF’

- **Java utiliza o padrão de caracteres Unicode, que abrange os conjuntos de caracteres de muitas línguas.**
- **Um literal de caracter é especificado por um único caracter entre apóstrofes simples: char exemplo = 'a';**

ELEMENTOS BÁSICOS DA SINTAXE
TIPOS/FORMATOS DE VARIÁVEIS

- **Tipo Booleano:**

TIPO	TAMANHO	ALCANCE
boolean	1 bit	true ou false

  - São variáveis lógicas que podem assumir os valores: verdadeiro e falso.
  - Note que, em Java, estas variáveis não podem ser interpretadas como os números inteiros 0 e 1.

ELEMENTOS BÁSICOS DA SINTAXE

CONSTANTE

- Na linguagem C, usa-se a palavra chave *const* para definir uma constante: const int x=100.

- Na linguagem Java, por sua vez, usa-se a palavra chave *final* com o mesmo propósito: final int x=100.

- Uma Variável é dita *final* quando seu valor não pode ser alterado após sua inicialização, ou seja, é uma Constante.

ELEMENTOS BÁSICOS DA SINTAXE

DECLARANDO VARIÁVEIS

- Declarando variáveis em Java:

```
public class VariableSamples {
public static void main( String[] args ){
        // declara uma variável com nome result e tipo boolean
            boolean result;
        // declara uma variável com nome option e tipo char
            char option;
        // atribui o símbolo C para a variável
            option = 'C';
        // declara uma variável com nome grade e tipo double
        // e a inicializa com o valor 0.0
            double grade = 0.0;
}
}
```

ELEMENTOS BÁSICOS DA SINTAXE

DECLARANDO VARIÁVEIS

- **Exibindo o valor de variáveis em Java:**

```
System.out.println()
System.out.print()

Aqui está um simples programa como exemplo:

public class OutputVariable {
        public static void main( String[] args ){
                int value = 10;
                char x;
                x = 'A';
                System.out.println(value);
                System.out.println("O valor de x = " + x );
        }
}

A saída deste programa será a seguinte:
        10
        O valor de  x = A
```

ELEMENTOS BÁSICOS DA SINTAXE

VARIÁVEIS - EXERCÍCIOS

- Levando em consideração as regras aprendidas para nomear as variáveis em Java, informar se os nomes abaixo estão corretos, justificando a resposta:
  1. int idade
  2. float a1b2c3
  3. float 7a6b5c
  4. char float
  5. char sim?não?
  6. int _alfa
  7. int alfa
  8. char num, Num

ELEMENTOS BÁSICOS DA SINTAXE

COMENTÁRIOS

- De única linha - //:
  - Um comentário curto relativo a uma linha de código pode ser incluído no fim da linha, precedido de //:
  -

```
..... ; // Comentário sobre o comando .....
```

- De uma ou mais linhas - /*.....*/:
  - Um comentário precedido de /* e seguido de */ pode ocupar várias linhas:
    -

```
/* Comentário maior,
que pode ocupar várias linhas. */
```

ELEMENTOS BÁSICOS DA SINTAXE

COMENTÁRIOS

- De documentação /** ..... */:
  - /** Indicam que o comentário deve ser inserido em qualquer documentação gerada automaticamente, por exemplo pelo javadoc . */
  - O JDK oferece um recurso para gerar automaticamente um arquivo HTML documentando uma classe.
  - Comentários que forem precedidos de /** e seguidos de */ serão incluídos neste arquivo.
  - Para gerar a documentação relativa à(s) classe(s) cuja(s) fonte(s) estão no arquivo MeuPrograma.java, digita-se na linha de comando: javadoc MeuPrograma.java

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- Uma expressão é uma instrução de realizar uma operação que produz um valor (valor de retorno).

- Operadores são símbolos especiais utilizados para operações matemáticas, atribuições, comparações e operações lógicas.

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores Aritméticos:**

OPERADOR	SIGNIFICADO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão (Módulo)

  - **Observação: Em Java não existe um operador específico para potência (tal como ** em Portugol). Para calcular uma potência, usa-se o método pow da classe Math do pacote lang, ou seja, para calcular x^y, se escreve Math.pow(x, y), onde x e y são de tipo double, e o resultado também**.

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores Aritméticos:**

TIPO DO RESULTADO	TIPOS DOS OPERANDOS
long	Nenhum operando é **float** ou **double** (aritmética de inteiros) e pelo menos um é **long**
int	Nenhum operando é **float**, **double** ou **long**
double	Pelo menos um operando é **double**
float	Pelo menos um operando é **float** e nenhum operando é double

- **Observação: Quando um inteiro e um real são usados como operandos em uma mesma operação aritmética, o resultado é um real.**

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores de Atribuição:**

OPERADOR	USO	DESCRIÇÃO
=	x = 5	Atribui o valor da direita à esquerda
+=	x += y	Equivalente a x = x + y
-=	x -= y	Equivalente a x = x - y
*=	x *= y	Equivalente a x = x * y
/=	x /= y	Equivalente a x = x / y
%=	x %= y	Equivalente a x = x % y

- **Observação: Estes operadores são simplesmente uma notação compacta para uma operação aritmética seguida da atribuição do valor de retorno à variável que continha o primeiro termo da operação.**

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores de Incremento e Decremento:**

OPERADOR	EXEMPLO	SIGNIFICADO
++	++a	Adicionar 1 à variável a e depois calcular a expressão na qual a reside
	a++	Calcular a expressão na qual a reside e depois adicionar 1 à variável a
--	--a	Subtrair 1 da variável a e depois calcular a expressão na qual a reside
	a--	Calcular a expressão na qual a reside e depois subtrair 1 da variável a

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- Operadores de Incremento e Decremento
- Exemplo:
- int m = 7;
- int n = 7;
- int a = 2 * ++m // agora a é 16, m é 8.
- int b = 2 * n++ // agora b é 14 e n é 8

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores de Comparação**

OPERADOR	SIGNIFICADO
==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

- **Estes operadores atuam sobre valores numéricos e retornam valores booleanos, true (verdadeiro) ou false (falso).**
- **O operador == também serve para comparar outros tipos de dados, inclusive objetos.**

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- ## Operadores Lógicos

OPERADOR	SIGNIFICADO	EXEMPLO	EXPLICAÇÃO
&&	E (“logical AND”)	a && b	Retorna true se a e b forem ambos true. Senão retorna false. Se a for false, b não é avaliada.
&	E ("boolean logical AND")	a & b	Retorna true se a e b forem ambos true. Senão retorna false. Ambas expressões a e b são sempre avaliadas.
||	OU ("logical OR")	a || b	Retorna true se a ou b for true. Senão retorna false. Se a for true, b não é avaliada.

  - Estes operadores atuam sobre valores booleanos e retornam valores booleanos, true ou false.

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Operadores de Comparação**

OPERADOR	SIGNIFICADO	EXEMPLO	EXPLICAÇÃO
|	OU ("boolean logical inclusive OR")	a | b	Retorna true se a ou b for true. Senão retorna false. Ambas expressões a e b são sempre avaliadas.
^	OU EXCLUSIVO ("boolean logical exclusive OR")	a ^ b	Retorna true se a for true e b for false ou vice-versa. Senão retorna false
!	NÃO ("logical NOT")	! a	Retorna true se a for false. Senão retorna false

  - Estes operadores atuam sobre valores booleanos e retornam valores booleanos - true ou false.

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Caracteres de Escape**

ESCAPE	SIGNIFICADO
\n	nova linha
\t	tabulação
\b	passo para trás
\r	retorno do carro
\\	barra invertida
\'	apóstrofe
\"	aspas

- Para caracteres de escape, usa-se a barra invertida.

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Precedência e associatividade dos operadores**

Operador	Associatividade
()	da esquerda para a direita
++ --	unários; da direita para a esquerda
* / %	da esquerda para a direita
+ -	da esquerda para a direita
< <= > >=	da esquerda para a direita
== !=	da esquerda para a direita

ELEMENTOS BÁSICOS DA SINTAXE
EXPRESSÕES E OPERADORES

- **Precedência e associatividade dos operadores**

Operador	Associatividade
&	da esquerda para a direita
^	da esquerda para a direita
|	da esquerda para a direita
&&	da esquerda para a direita
||	da esquerda para a direita
= += -= *= /= %=	da direita para a esquerda

ELEMENTOS BÁSICOS DA SINTAXE

STRINGS

- String: São sequências de caracteres - na linguagem Java temos a uma classe pré-definida chamada String na biblioteca padrão.
  - Exemplo:
  - String e = “ ”; // string vazia
  - String greeting = “Ola”;

ELEMENTOS BÁSICOS DA SINTAXE

STRINGS

- Substring: Podemos extrair parte de uma string maior com o método substring da classe string.
  - Exemplo:
  - String greeting = “Ola”;
  - String s = greeting.substring(0, 2);
    - Criando uma string chamada “Ol”.

ELEMENTOS BÁSICOS DA SINTAXE

STRINGS

- Strings são imutáveis, ou seja, não existe nenhum método que permite alteração de caractere em uma string já existente.
- Mas podemos alterar as posições, de uma forma fácil.
  - Exemplo: Vamos alterar a nossa string de “Ola” para “Ole”.
  - greeting = greeting.substring(0,2) + “e”;
  - Nossa string vai ficar = “Ole”.

MEU PRIMEIRO PROGRAMA EM JAVA

- Já se tornou clássica a ideia de que para aprender uma nova linguagem de programação não se deve ir direto à sua descrição formal.
- Ao invés disso, é melhor examinar cuidadosamente um pequeno programa escrito nessa linguagem, o mais simples possível, mas que permita "quebrar o gelo".

MEU PRIMEIRO PROGRAMA EM JAVA

- Isso faz sentido pois, por exemplo, quando vamos aprender Inglês, ou outro idioma qualquer, não iniciamos com a leitura compenetrada de um livro de gramática, mas aprendemos algumas estruturas simples e procuramos exercitá-las, adiando o estudo rigoroso para quando estivermos suficientemente maduros.

- Ao compreender as diversas partes componentes do exemplo, já teremos dado um grande passo para podermos escrever qualquer programa.

MEU PRIMEIRO PROGRAMA EM JAVA

- Seguindo essa linha, apresentamos nosso primeiro programa, o clássico "Alo pessoal!".
- O objetivo deste programa é simplesmente escrever na tela a frase "Alo pessoal!".
- Vejamos como é o código fonte:

MEU PRIMEIRO PROGRAMA EM JAVA


```
public class AloPessoal {
    public static void main(String args[ ]) {
        System.out.println("Alo pessoal!");
}
}
```

MEU PRIMEIRO PROGRAMA EM JAVA

Digitando o Programa:

- Que tal colocarmos a mão na massa e digitarmos esse programa?
- Para isso é recomendável utilizar um editor de texto simples como o Notepad ou Bloco de Notas do Windows, ou ainda os ambientes como o Netbeans ou Eclipse.

MEU PRIMEIRO PROGRAMA EM JAVA

- Se estiver no Unix, use preferencialmente o TextEdit ou o vi.
- O nome do arquivo deve ser exatamente igual ao nome que aparece após a palavra class na primeira linha do programa e dever ter .java como sufixo.
- Cuidado para digitar corretamente as maiúsculas e minúsculas, pois a linguagem Java é sensível ao tipo de caixa das letras.

MEU PRIMEIRO PROGRAMA EM JAVA

- Compilando o Código Fonte:
- Para criar o código binário, chamamos o compilador Java através da linha de comando, do seguinte modo:

javac AloPessoal.java

- Com isso, será criado um arquivo binário (desde que tudo corra bem) com mesmo nome do arquivo original, mas com sufixo .class no lugar de .java.
- No nosso caso, teríamos um arquivo AloPessoal.class.

MEU PRIMEIRO PROGRAMA EM JAVA

Compilando o Código Fonte:

- Entre as (muitas) coisas que poderiam dar errado nesse momento, o código fonte pode conter erros.
- Esse não deverá ser o caso, se tiver digitado o programa exatamente como descrito.
- Se, porém, o compilador emitir mensagens de erro, será preciso identificar as linhas que estão incorretas, corrigi-las no editor, e chamar o compilador novamente.
- As linhas que contém algum tipo de incorreção são listadas pelo compilador juntamente com seu número, facilitando sua localização no editor.

MEU PRIMEIRO PROGRAMA EM JAVA

Executando o Código:

- Para podermos executar o programa é necessário chamar o interpretador Java, pois, como vimos, os bytecodes foram feitos para rodar em uma *Java Virtual Machine*.
- Podemos fazê-lo do seguinte modo:

  java [nome da classe]

- Onde [nome da classe] é o nome do arquivo sem o sufixo .class.
- Em nosso caso, este será AloPessoal.

MEU PRIMEIRO PROGRAMA EM JAVA

Entendendo a Estrutura do Programa:

- Todo programa Java, deve conter ao menos uma declaração da forma:

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

- Onde [nome] é o nome da Classe e a parte "..." é o código Java válido, a ser executado no programa.

MEU PRIMEIRO PROGRAMA EM JAVA

Entendendo a Estrutura do Programa:

- **O nome de uma Classe é um identificador, como qualquer outro presente no programa, por isso não deve conter espaços ou outros caracteres gráficos, isto é, deve ser um nome composto de uma sequência de caracteres que seja válida para um identificador.**
- **Outros exemplos de identificadores são nomes de variáveis, nomes de comandos, etc.**
- **Vamos adiar um pouco a complicação sobre o que vem a ser uma Classe, pois isso depende de alguns conceitos da programação orientada a objetos.**

MEU PRIMEIRO PROGRAMA EM JAVA

Entendendo a Estrutura do Programa:

- **Por hora, vamos apenas aceitar que todo programa Java deve conter ao menos uma Classe, e que é dentro de uma Classe que vão os dados e os procedimentos.**

- **Notemos ainda que todo programa Java (mas não as applets – executado e visualizado dentro de uma página HTML) deve ter uma Classe dotada de um procedimento chamado main.**

MEU PRIMEIRO PROGRAMA EM JAVA

Entendendo a Estrutura do Programa:

- Os procedimentos em Java são chamados Métodos.
- Os Métodos encerram um conjunto de declarações de dados e de comandos que são executados mediante a chamada do método por seu nome.
- O Método main é o ponto onde se dá o início da execução do programa, isto é, um Método é chamado automaticamente pela JVM.

MEU PRIMEIRO PROGRAMA EM JAVA

Executando o Código:

- **Ao executar o programa, ele deverá exibir na tela a frase:**

  Alo pessoal!

- **Se isto não ocorrer, volte atrás e verifique se o programa foi digitado exatamente como aparece na listagem acima.**

- **Este é um programa muito simples, porém, a boa compreensão das estruturas presentes nele deverá permitir a programação fluente em Java em pouco tempo.**

COMANDOS DE SAÍDA DE INFORMAÇÕES

- Podemos apresentar a saída de informações em Java, através dos comandos:
  - System.out.println
  - System.out.print
  - System.out.printf

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.println
  - “System.out” - Objeto de saída padrão.
  - “println” - Método que imprime uma string (conjunto de caracteres) e passa para a próxima linha.
  - Exemplo: System.out.println(“Bem vindo a FMU”;

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.print
  - “System.out” - é um objeto de saída padrão.
  - “print” - é um método que imprime uma mensagem e não passa para a próxima linha.
  - Exemplo: System.out.print(“Bem-vindo a “;
    System.out.println(“FMU”;

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.printf
  - “System.out” - Objeto de saída padrão.
  - “printf” - Método que imprime a mensagem com argumentos, separados por vírgulas.
  - Exemplo: System.out.printf("%s%s", "Bem vindo ", "a FMU”;

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.printf
  - O primeiro argumento contém os especificadores de formato (iniciados por “%”).
  - Nesse caso, “%s” significa que vai ser formatado uma string.
  - Repare que existem dois “%s” na especificação de formato e outros dois argumentos após ele, indicando o que vai aparecer na tela.

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.printf
  - É necessário que haja um casamento entre a quantidade de especificadores de formato e de argumentos após a string de argumentos.
  - Existem outros especificadores de formato, como para números inteiros (%d) e reais (%f).

COMANDOS DE SAÍDA DE INFORMAÇÕES

- System.out.printf
  - Outro exemplo:

    System.out.printf("Nome: %s\nIdade: %d\nAltura: %.2f\n", "Maria", 19, 1.69);

  - Os tipos dos especificadores devem coincidir com os tipos e posições dos argumentos, ou seja, por exemplo "%s" com "Idade: ", "%d" com 19 e "%f" com 1.69.

COMANDOS DE SAÍDA DE INFORMAÇÕES

CÓDIGO DO FORMATO	SIGNIFICADO
%c	Apresenta um único caracter
%s	Apresenta uma cadeia de caracteres (string)
%d	Apresenta um número inteiro da base 10 (decimal)
%f	Apresenta um número de ponto flutuante (real)

COMANDOS DE SAÍDA DE INFORMAÇÕES EXERCÍCIOS

1. Uma prática comum entre programadores é que o seu primeiro programa em uma linguagem de programação deve dizer olá para o mundo, em inglês “Hello world!”. Escreva um programa que escreva essa mensagem, lembrando-se de coincidir o nome do arquivo com o nome da classe. Faça, pelo menos, três versões desse programa, para usar os métodos “System.out.print”, “System.out.println” e “System.out.printf”.

2. Escreva um programa que escreva os números de 1 a 10 em uma linha e de 11 a 20 na linha seguinte. Os números devem ter um espaço de alguns caracteres entre eles e os de baixo devem estar alinhados com os de cima. Dica: use a sequência de escape “\t” para que os números se posicionem em marcas de tabulação pré-definidas.

COMANDOS DE SAÍDA DE INFORMAÇÕES EXERCÍCIOS

3. No programa anterior, os números de baixo foram alinhados aos de cima pela esquerda, quando o comum para números é o alinhamento pela direita. Escreva um programa que faça o alinhamento pela direita. Dica: Use o especificador de formato "%d" indicando a quantidade de dígitos que o numero deve ter que você obterá o alinhamento pela direita.

4. Faça um programa que escreva a seguinte operação matemática:

```
123
+  8
-----
131
```

COMANDOS DE SAÍDA DE INFORMAÇÕES EXERCÍCIOS

5. Elabore um programa que mostre o seguinte menu na tela:

Cadastro de Clientes

0 - Fim
1 - Inclui
2 - Altera
3 - Exclui
4 - Consulta

Opção:

COMANDO DE ENTRADA DE DADOS

- As entradas são utilizadas para realizar interação com usuários, ou sistemas através de recebimento de dados ou arquivos.
- Para receber dados utilizaremos a classe Scanner.
- Para isso precisamos importar um pacote para utilizar a classe, dessa forma precisamos entender o conceito de API.

COMANDO DE ENTRADA DE DADOS

- API - Application Programming Interface:
  - Em Java, temos milhares de funcionalidades/classes.
  - Essas classes foram agrupadas em pacotes e os pacotes para a mesma funcionalidade são as APIs.
  - Por conta de performance dos programas, esses pacotes não ficam disponíveis.

COMANDO DE ENTRADA DE DADOS

- API - Application Programming Interface:
  - Para utilizar os pacotes e classes precisamos importá-los para os nossos programas.
  - Para utilizar a classe Scanner, precisamos importar para o programa, o seu pacote chamado de “java.util”.
  - Observação: Para os comandos de saída (println, print e printf) não precisamos importar nenhuma classe, pois são considerados métodos comuns.

COMANDO DE ENTRADA DE DADOS

- Para isso vamos adicionar o comando no início do programa:

```
Import java.util.Scanner;
```

- Agora vamos declarar nosso objeto do tipo scanner, chamado de “entrada”:

```
Scanner entrada = new Scanner(System.in);
```

- Agora o objeto está pronto para ler a entrada que o usuário vai digitar, atribuindo esse valor à variável definida previamente:

```
variável = entrada.nextInt();
```

COMANDO DE ENTRADA DE DADOS

- Segue exemplo de programa que solicita e mostra a idade de uma pessoa:

```
import java.util.Scanner;

public class Entrada {

    public static void main(String[ ] args) {
        Scanner entrada = new Scanner(System.in);
        int idade;

        System.out.println("Digite sua idade: ");
        idade = entrada.nextInt();

        System.out.print("Sua idade é: " + idade + "\n");

    }
}
```

COMANDO DE ENTRADA DE DADOS

- Vamos aprimorar, informando o ano de nascimento.
- Para isso precisamos importar a classe “Calendar” que possui métodos para trabalharmos com data e hora.
- Usaremos o método get(Calendar.YEAR), que retorna um inteiro com o ano, e vamos armazenar esse inteiro em uma variável 'ano_atual'.

COMANDO DE ENTRADA DE DADOS

```
import java.util.Scanner;
import java.util.Calendar;

public class Entrada {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int idade;
        int ano_atual;
        int ano_nascimento;

       System.out.println("Digite sua idade: ");    // Pergunta a idade
        idade = entrada.nextInt();    // Armazena a idade

        Calendar calendario = Calendar.getInstance(); // Criando o objeto “calendario” do tipo
Calendar
        ano_atual = calendario.get(Calendar.YEAR); ();    // Armazenando o ano atual

        ano_nascimento = ano_atual - idade;

        System.out.printf("Você nasceu em " + ano_nascimento + "\n");
    }
}
```

COMANDO DE ENTRADA DE DADOS

- Usamos o método get(Calendar.YEAR), que retorna um valor inteiro com o ano, e armazenamos esse inteiro na variável 'ano'.
- O ano de nascimento é calculado e armazenado através da operação de subtração: ano_nascimento = ano_atual - idade;
- Fizemos assim por ser um tutorial básico e também por questão de organização, mas a variável 'ano_nascimento' não seria necessária.

COMANDO DE ENTRADA DE DADOS

- Poderíamos ter usado '(ano_atual - idade)' direto no printf assim:

```
System.out.printf("Você nasceu em " + (ano_atual - idade) + "\n");
```

- Mas tem que ser entre parênteses.

COMANDO DE ENTRADA DE DADOS

- Assim, não haveria a necessidade da variável 'ano_nascimento'.
- Aliás, também não precisaríamos da variável 'ano_atual‘.
- Poderíamos ter feito diretamente assim:

```
System.out.printf("Você nasceu em " + ( calendario.get
(Calendar.YEAR) - idade) + "\n");
```

- Também podemos não ter a variável 'idade‘.
- Faça só:

COMANDO DE ENTRADA DE DADOS

```
import java.util.Scanner;
import java.util.Calendar;

public class Entrada {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        System.out.println("Digite sua idade: ");
        Calendar calendario = Calendar.getInstance();

        System.out.printf("Você nasceu em " +
(calendario.get(Calendar.YEAR) - entrada.nextInt()) + "\n");

    }
}
```

COMANDO DE ENTRADA DE DADOS

- Cada vez que tiramos as variáveis, o programa fica menor e ocupa menos espaço de memória, porém perde em legibilidade.

- Com 'idade', 'ano_atual' e 'ano_nascimento‘, o programa fica bem mais organizado.

ESTRUTURA CONDICIONAL SIMPLES

- Utilizamos o comando if para tomadas de decisões simples, utilizando um comando alternativo.

- A sintaxe do comando, segue a estrutura abaixo:

```
If (condição) {
        instrução 1;
        instrução 2;
}
```

- Se a condição/expressão for verdadeira, o bloco de instruções será executado; caso contrário, nada será feito.

ESTRUTURA CONDICIONAL SIMPLES

- Exemplo: Comparação entre dois números - vamos testar a condição '1 é igual a 2', que retornará 'false' e depois testar se ‘1 é igual 1’.

```
public class If {
    public static void main(String[ ] args) {
        if (1 == 2){
            System.out.println(“Essa mensagem nunca pode
            aparecer, se está lendo algo deu errado");
        }
        if (1 == 1){
            System.out.println("1 é igual a 1, com certeza");
        }
    }
}
```

ESTRUTURA CONDICIONAL COMPOSTA

- A estrutura condicional ou de decisão composta serve para escolher um entre dois ou mais comandos alternativos.

- Utilizamos o comando if - else para tomada de decisão composta.

ESTRUTURA CONDICIONAL COMPOSTA

- A sintaxe do comando, segue a estrutura abaixo:

```
if (condição) {
        instrução 1;
} else {
        instrução 2;
}
```

- Funcionamento:
  - A condição é avaliada; se o resultado for verdadeiro, é executada a instrução 1 e se o resultado for falso, executa-se a instrução 2.

ESTRUTURA CONDICIONAL COMPOSTA

- **Exemplo: Criar um programa que receba uma nota (classe Scanner) e valida se você passou direto, ficou de recuperação ou foi reprovado na matéria, exibindo tal mensagem. Regras de validação:**

  Nota 7 ou mais: Parabéns, você passou direto.

  Entre 4 e 7: Você ficou de exame.

  Abaixo de 4: Você foi reprovado.

ESTRUTURA CONDICIONAL COMPOSTA

- Primeiro precisamos garantir que será digitada uma nota válida, ou seja, entre 0 e 10. Assim, a primeira parte de nosso programa ficará assim:

```
public class Ifelse {
    public static void main(String[] args) {
        float nota; //vai armazenar a nota
        Scanner entrada = new Scanner(System.in);

        System.out.print("Digite sua nota [0.0 - 10.0]: " );
        nota = entrada.nextFloat();

        if ( (nota <= 10.0) && (nota >= 0.0) ) {
            System.out.println("Nota válida");
        } else {
            System.out.println("Nota inválida, fechando o aplicativo");
        }
    }
}
```

ESTRUTURA CONDICIONAL COMPOSTA

- Agora faremos a próxima validação, ou seja, se o aluno passou direto - se o aluno tirou 7 ou mais.

```
If ( nota >= 7.0 ) {
        System.out.println("Parabéns, você passou direto");
}
else {
        System.out.println("Você não passou direto");
}
```

ESTRUTURA CONDICIONAL COMPOSTA

- **Unificando com nosso código:**

```
public class Ifelse {
   public static void main(String[] args) {
      float nota; //vai armazenar a nota
      Scanner entrada = new Scanner(System.in);

      System.out.print("Digite sua nota [0.0 - 10.0]: " );
      nota = entrada.nextFloat();

      if ( (nota <= 10.0) && (nota >= 0.0) ) {
         System.out.println("Nota válida");

         if ( nota >= 7.0 ) {
            System.out.println("Parabéns, você passou direto");
         }
         else {
            System.out.println("Você não passou direto");
         }

      }
      else {
         System.out.println("Nota inválida, fechando o aplicativo");
      }

   }
}
```

ESTRUTURA CONDICIONAL COMPOSTA

- A próxima validação é se o aluno ficou de exame, ou seja se a nota ficou entre 5 e 7.

```
If (nota >= 5.0) {
        System.out.println(“Você ficou de Exame");
}
else {
        System.out.println(“Você foi Reprovado");
}
```

- Note que se não for maior ou igual a 5.0, é menor, então vai para o else - se é menor, foi reprovado.

ESTRUTURA CONDICIONAL COMPOSTA

- Então o else manda mensagem de reprovação.
- Onde vamos inserir esse código?
- Resposta: No else do trecho “Você não passou direto”.

ESTRUTURA CONDICIONAL COMPOSTA

- O código completo ficaria assim:

```
import java.util.Scanner;

public class Ifelse {
   public static void main(String[] args) {
      float nota; //vai armazenar a nota
      Scanner entrada = new Scanner(System.in);

      System.out.print("Digite sua nota [0.0 - 10.0]: " );
      nota = entrada.nextFloat();

      if( (nota >=0.0) && (nota <= 10.0) ) {

         if ( nota >= 7.0 ) {
            System.out.println("Parabéns, você passou direto.");
         }
         else {

            if ( nota >= 5.0 ) {
               System.out.println(“Você ficou de exame");
            }
            else {
               System.out.println(“Você foi reprovado.");
            }
         }

      }
      else {
         System.out.println("Nota inválida, fechando aplicativo");
      }
   }
}
```

ESTRUTURA CONDICIONAL COMPOSTA

- O Java lê e interpreta seguindo esses passos:
  - Primeiro avalia se a nota é válida (se está entra 0 e 10)
  - Se não está, vai para o else, que termina o programa.
  - Se está, vai para o próximo if, que é o passo 2.
  - Avalia se tirou mais que 7.0 através do próximo if
  - Se sim, ele passa direto e termina o programa.
  - Se não tirou mais que 7.0, vai pro else, que é o passo 3.
  - Avalia se tirou mais que 5.0 através do próximo if
  - Se sim, diz que ficou de recuperação e termina o programa.
  - Se não, vai pro else, que diz que foi reprovado e terminou o programa.
- Por questão de organização, coloque os pares if - else na mesma linha vertical, veja como fica mais organizado, e assim você vai saber à qual if o else pertence (sim, você pode se confundir).

ESTRUTURA CONDICIONAL
EXERCÍCIOS

1. Fazer um programa que leia um número inteiro qualquer e mostre se ele é par ou impar.

2. Fazer um programa que leia um número inteiro qualquer e mostre se ele é positivo, negativo ou nulo.

3. Desenvolver um programa em Java que leia um número inteiro e valide se ele está entre 0 e 10. Se válido, apresentar o valor na tela.

ESTRUTURA CONDICIONAL
EXERCÍCIOS

4. Desenvolver um programa em Java simulando os dias da semana, ou seja, o usuário deve digitar um número entre 1 e 7, onde cada número corresponde a um dia da semana. Exemplo: 1 = Domingo ... 7 = Sábado.

5. Tendo como dados de entrada: a altura e o sexo de uma pessoa, construa um programa que calcule e mostre o seu peso ideal, utilizando as fórmulas:

   Sexo Masculino: Peso ideal = (72,7 X Altura) - 58.
   Sexo Feminino: Peso ideal = (62,1 X Altura) - 44,7.

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

- As instruções switch - case - break - default são utilizadas em tomadas de decisão onde o número de possibilidades é grande, normalmente maior que duas, senão utilizamos o if - else.

- Utilizamos para diminuir a complexidade do if - else encadeado.

- Colocamos várias opções e vários comandos dentro do comando switch, todas as possibilidades de nosso aplicativo ou todas as opções ou rumos que nossos programas possam tomar.

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

A sintaxe:

```
switch (condição) {
        case constante1:
                instruções1;
                break;
        case constante2:
                instruções2;
                 break;
        ..............
         case constanten:
                instruçõesn;
                 break;
        default:
                 instruções;
}
```

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

Funcionamento:

- **A condição é avaliada.**
- **Se o valor for igual a alguma das constantes que seguem o case, então são executadas as instruções que seguem o referido case.**
- **Se o valor da expressão não for igual a nenhuma das constantes apresentadas pelos case, então são executadas as instruções que seguem o default.**

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

Funcionamento:

- **O default é opcional. No caso de o valor da expressão não ser igual a nenhum dos case, nada é executado terminando o switch.**
- **O programa continua na instrução seguinte do switch.**
- **Importante: Em cada case do switch só uma única constante pode estar associada ao case.**

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

Funcionamento:

- **Se executarmos o programa sem o break, ele terá um comportamento estranho, testando todas as possibilidades dos case, mesmo que tenha sido atendida uma das condições.**

- **Para que isso não ocorra, utilizamos a instrução break, que permite parar a execução dentro de um switch, continuando o programa a partir da próxima instrução.**

- **O switch é mais eficiente que um encadeamento de if - else - if.**

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

- **Exemplo: Escreva um programa que indique qual o estado civil correspondente a um caracter escrito em maiúscula ou minúscula, a saber:**

  C - Casado

  S - Solteiro

  D - Divorciado

  V - Viuvo

  caso contrário, Estado civil incorreto.

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA

```
import java.util.Scanner;
 public class EstadoCivil {
 public static void main(String args[])
 {
  char Est_civil;
      Scanner entrada = new Scanner(System.in);
            System.out.print("Digite seu estado civil: ");
      Est_civil = entrada.nextLine().charAt(0);
      switch( Est_civil )
      {
         case 'C': case 'c':
            System.out.println("Casado");
            break;
         case 'S': case 's':
            System.out.println("Solteiro");
            break;
         case 'D': case 'd':
            System.out.println("Divorciado");
            break;
         case 'V': case 'v':
            System.out.println("Viuvo");
            break;
         default:
            System.out.println("Estado civil incorreto");
      }
 }
 }
```

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA
EXERCÍCIOS

1. Fazer um programa em Java que leia um Código e identifique e mostre a Opção escolhida, a saber: 1 - Inclusão / 2 - Alteração / 3 - Exclusão, senão Opção Inválida.

2. Desenvolver um programa em Java que, dados dois números e o sinal da operação, realize uma das 4 operações básicas (adição, subtração, multiplicação e divisão), utilizando switch e case..

3. Desenvolver um programa em Java que receba um número, referente ao mês do ano, e informe quantos dias aquele mês possui.

ESTRUTURA SELEÇÃO DE MÚLTIPLA ESCOLHA EXERCÍCIOS

4. Fazer um programa que, dado a idade de um nadador, classifique-o nas categorias: até 4 anos - Fraldinha, até 10 anos - Infantil, até 17 anos - Juvenil, e acima de 17 anos - Adulto.

5. Elaborar um programa que calcula o novo salário de um funcionário de uma empresa, reajustado de acordo com a sua função, a saber: Função = 1: 50%; Função = 2: 50%; Função = 3: 30%; Função = 4: 20%; Demais Funções: 10%.

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

- A instrução while executa uma instrução ou bloco de instrução, enquanto sua condição for verdadeira.
- Essa estrutura de repetição também é chamada de laço ou loop.
- A função ou o conjunto de instruções de um laço / loop, também é conhecida como corpo do laço.

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Sintaxe:

```
while (condição) {
        Comando_1;
        Comando_2:
        ...
        Comando_n;
}
```

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Funcionamento:

- **A condição é avaliada.**
- **O resultado sendo falso (zero), o laço termina e o programa continua a partir da próxima instrução seguinte do while.**
- **O resultado sendo verdadeiro (diferente de zero), é executada a instrução ou bloco de instruções associadas ao while.**

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Exemplo 1: Desenvolver um programa em Java que apresente na tela os números entre 1 e 10.

```
public class Contando {
    public static void main(String[ ] args) {
        int count = 1;

        while (count<=10) {
            System.out.println(count);
            count++;
        }
    }
}
```

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Exemplo 2: Desenvolver um programa em Java que apresente na tela os números entre 100 e 1.

```
public class Contando {
    public static void main(String[ ] args) {
        int count = 100;

        while (count >= 1) {
            System.out.println (count);
            count--;
        }
    }
}
```

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito:

- **Podemos desenvolver um laço que não pare de rodar, enquanto a condição for verdadeira.**
- **Utilizado muitas vezes em pesquisas acadêmicas, testes de capacidade, validação de performance, ou seja temos muitas utilizações.**
- **Ou seja, enquanto ó código for verdadeiro permanece sendo executado.**

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito - Exemplo 1:

```
public class Loop {
    public static void main(String[ ] args) {
        while (true) {
            System.out.println("Loop infinito!");
        }
    }
}
```

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito - Exemplo 2: Desenvolver um programa que permaneça rodando, enquanto o usuário não descobrir o código chave, Esse código deve ser um único caracter.

```
import java.util.Scanner;

public class Matrix {
   public static void main(String[] args) {
      boolean continuar = true;
      char opcao;
      Scanner entrada = new Scanner(System.in);

      while(continuar){
         System.out.println("Você está na matrix;");
         System.out.print("Digite o caracter especial para sair da matrix: ");
         opcao = entrada.next().charAt(0);

         if(opcao==‘j'){
            continuar=false;
            System.out.println("Parabéns! Você conseguiu sair da Matrix!");
         }
         else{
            System.out.println("Você não está autorizado a sair da Matrix. Estude Java.");
         };
      }
   }
}
```

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito

- No exemplo 1 de looping, a condição é sempre verdadeira. Aí não tem muito o que fazer. Ela foi definida assim e vai continuar assim.
- Em Java, como em outras linguagens, não usamos esse operador booleano diretamente.
- Ao invés disso, declaramos uma variável do tipo 'boolean' que vai definir se o laço deve continuar ou não.
- E como isso vai ser decidido? Por você, durante a execução do programa.

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito

- No exemplo 2, foi criada uma variável 'boolean' chamada 'continuar'. Inicialmente ela é 'true' para o while começar.
- Se ela tivesse sido iniciada como 'false‘, o laço nem iniciaria e não existiria programa.
- Declaramos um tipo char 'opcao' que vai receber um caracter do usuário. Vai ser tipo uma senha. Só vai escapar do while se o usuário digitar essa senha, ou caracter.
- Muito bem, o programa inicia e diz que para você sair da Matrix precisará digitar um caracter especial.
- Como você é programador Java, está vendo que no teste condicional a senha é ‘j', de java.

ESTRUTURA REPETIÇÃO COM TESTE NO INÍCIO

Loop Infinito

- Se o usuário adivinhar a letra, o if retorna verdadeiro e altera o valor de 'continuar' para false, aí na próxima iteração o while não irá executar e o programa sai do loop.
- Caso ele erre - o que é provável, visto que existem 26 letras e 10 números no teclado - a variável 'continuar', continua sendo true (pois nada foi alterado).
- É o seu primeiro programa de 'segurança' em Java, pois exige uma senha.
- É possível transformar em executável ou inviabilizar que o usuário não veja essa senha, assim só o programador sabe a senha de acesso.
- Você pode alterar de 'j' para a inicial de seu nome.

ESTRUTURA REPETIÇÃO COM TESTE NO FINAL

- A instrução do...while, diferente dos outros laços, pois o teste é feito no fim da instrução do laço e não antes, como no for e while.

- Dessa forma o corpo do laço do...while é executado ao menos uma vez, enquanto nos laços while e for, o corpo do laço pode nunca ser executado, caso a condição seja sempre falsa.

ESTRUTURA REPETIÇÃO COM TESTE NO FINAL

Sintaxe:

```
do {
      Comando_1;
      Comando_2;
      .....
      Comando_n;
} while (condição);
```

Funcionamento:

- A instrução é executada.
- A condição é avaliada.
- Se a condição for verdadeira, retorna ao ponto inicio.
- Se a condição for falsa, termina o laço e continua a partir da próxima instrução.
- O laço do...while está adaptado ao processamento de menus.

ESTRUTURA REPETIÇÃO COM TESTE NO FINAL

Exemplo:

```
import java.util.Scanner;

public class DoWhile {
   public static void main(String[] args) {
      boolean continuar=true;
      int opcao;
      Scanner entrada = new Scanner(System.in);
      do
      {
         System.out.println("\t\tMenu de opções do curso Java Progressivo:");
         System.out.println("\t1. Ver o menu");
         System.out.println("\t2. Ler o menu");
         System.out.println("\t3. Repetir o menu");
         System.out.println("\t4. Tudo de novo");
         System.out.println("\t5. Não li, pode repetir?");
         System.out.println("\t0. Sair");

         System.out.print("\nInsira sua opção: ");
         opcao = entrada.nextInt();

         if(opcao == 0){
            continuar = false;
            System.out.println("Programa finalizado.");
         }
         else{
            System.out.printf("\n\n\n\n\n\n\n");
         }
      } while( continuar );
   }
}
```

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

- A instrução for é utilizada mais comumente em situações onde o número de iterações é conhecido.

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

Sintaxe:

```
for (inicialização; condição; incremento) {
        Comando_1;
        Comando_2;
         .....
         Comando_n;
}
```

Onde:

- **Inicialização: Declaração usada para inicializar a variável de controle do laço.**
- **Condição: Expressão relacional que determina quando o laço terminará pelo teste da Variável de Controle.**
- **Incremento: Define como a variável de controle do laço mudará, cada vez que a repetição for realizada.**

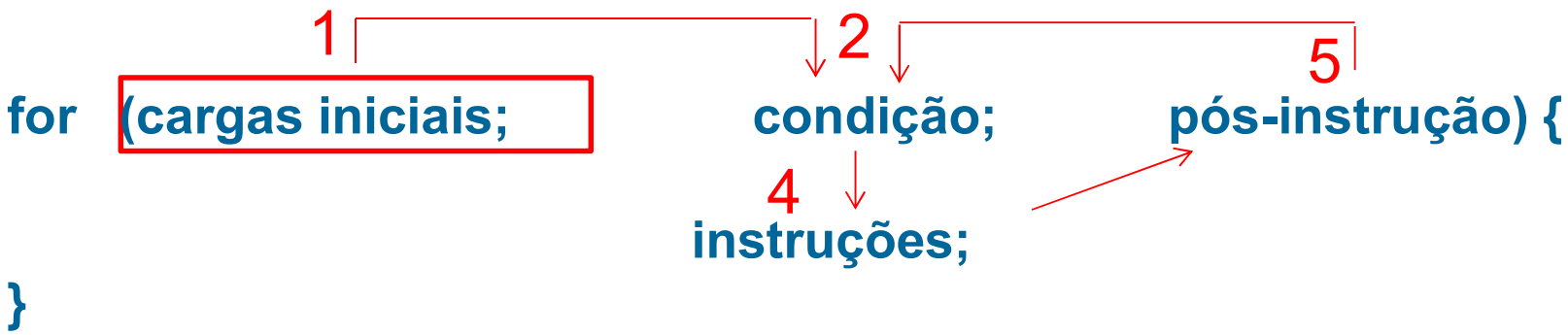
ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

- A instrução for é utilizada mais comumente em situações onde o número de iterações é conhecido.

- Apresenta um formato estranho, mas é um laço bem desenhado, que consegue resumir em uma mesma instrução repetitiva, tudo o que ela precisa.

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

Funcionamento:



ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

Funcionamento:

1. **Código presente em cargas iniciais é executado. Aqui são iniciadas as varáveis presentes no laço. Esse componente do laço for é executado apenas uma vez.**
2. **A condição é avaliada.**
3. **Se o resultado da condição retorna falso, o laço termina e o programa continua na instrução a seguir do loop.**
4. **Se o resultado da condição retorna verdadeiro, então são executadas as instruções do loop.**
5. **Após a execução das instruções, é executada a pós-instrução. Nesse componente do laço for, são realizadas as alterações necessárias para passar a próxima iteração do loop.**
6. **Voltar ao ponto 2.**

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

- Exemplo1: Vamos retornar ao exemplo do while - como sabemos que o número de vezes que o programa irá iterar o laço é de 10 vezes, podemos reescrevê-lo utilizando a estrutura for.
- Informações necessárias para criar o laço:
  - Iniciar a variável de controle do laço.
  - A condição se mantém a mesma.
  - Escrever o número na tela.
  - Para passar para a próxima iteração, precisamos incrementar a variável de controle do laço em uma unidade. Ficando da seguinte forma:

```
public class for1 {
    public static void main(String[] args) {
        for(int count=1 ; count <= 10 ; count++){
            System.out.println(count);
        }
    }

}
```

```
int count=1;
        while(count<=10) {
            System.out.println(count);
            count++;
        }
```

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

- Exemplo 2: Desenvolver um programa em Java que apresente na tela os números entre 100 e 1.

```
public class for2{
    public static void main(String[] args) {
        for(int count=100 ; count >= 1; count--){
            System.out.println(count);
        }
    }
}
```

ESTRUTURA REPETIÇÃO COM VARIÁVEL DE CONTROLE

Exemplo 3: Desenvolver um programa em Java que apresente na tela os números entre 1 a 10 em uma coluna e na outra 10 a 1.

```
public class for3 {
    public static void main(String[] args) {
        for(int sobe=1, desce=10 ; sobe<=10 && desce>=1; sobe++, desce--){
            System.out.printf("%d \t %d \n", sobe, desce);
        }
    }
}
```

RESUMO DAS ESTRUTURAS DE REPETIÇÃO

	While	For	Do...While

Sintaxe	while (cond){
	instruções;
	}
		for (carga inic.;
		cond; pós-inst) {
		instruções;
		}
			do {
			instruções;
			} while condição);

Executa a instrução	Zero ou mais vezes	Zero ou mais vezes	1 ou mais vezes
Testa a condição	Antes da instrução	Antes da instrução	Após a instrução
Utilização	Frequente	Frequente	Pouco frequente

ESTRUTURAS DE REPETIÇÃO
EXERCÍCIOS

1. Elaborar um programa em Java que calcule e mostre a soma de uma série de números inteiros, sendo o último deles = -1 - FLAG, não considerado no processo.

2. Elaborar um programa em Java que calcule e mostre a média aritmética das idade dos alunos do sexo feminino, de uma sala de 50 alunos.

3. Elaborar um programa em Java que mostre os números ímpares existentes entre os números inteiros de 1 até 500.

4. Desenvolver um programa em Java, gera 20 números entre 1000 e 1999 e mostra os que deixam resto 5 na divisão por 11.

5. Desenvolver um programa em Java utilizando While e do While para realizar uma contagem regressiva de um número digitado pelo usuário.

Biblioteca Virtual

- http://portal.fmu.br/
- https://acessobiblioteca.fmu.br/

Bibliografia

- ASCENCIO, Ana Fernanda Gomes. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ (padrão ANSI) e Java. & CAMPOS, Edilene Aparecida Veneruchi de. 3ª Edição. São Paulo: Pearson Education do Brasil, 2012. (Disponível na Biblioteca Virtual 3.0)

- DEITEL, Paul; DEITEL, Harvey. Java, Como Programar [recurso eletrônico, Biblioteca Virtual Universitária 3.0]. 8ª ed. São Paulo : Pearson Prentice Hall Brasil, 2010.