

POO I

Vinícius Machado

Sobre o professor

- Colégio Técnico Industrial Profº Mário Alquati, hoje IFRS.
- FURG - Engenharia de Computação.
 - Bolsista PET-C3
 - TCC Realidade Aumentada
 - Professor de cursinho
- UFRGS - Mestrado em Ciências da Computação.
 - Professor de cursinho
 - Professor na QI
- Knewin - Desenvolvedor abril/2016 até dez/2017
- IFRS (desde abril/2017)

Sobre a disciplina

Encontros

Terças-feiras 19:05 até 22:40

Materiais e códigos desenvolvidos em sala de aula disponíveis via GITHUB

<https://github.com/vfmachado/POO-2020/>

Ementa

Conceitos de Orientação a Objetos. Mecanismos de Abstração e Composição. Construções de linguagens orientadas a objeto: classes, objetos, instâncias, atributos e métodos. Modelo de execução de um programa orientado a objeto. Estado e Comportamento. Encapsulamento e Ocultamento de Informação. Associação, Agregação e Composição. Herança e Polimorfismo. Classes abstratas. Interfaces. Classes Internas. Tipos Genéricos. Tratamento de erros. Organização em Pacotes e Camadas.

Cronograma

1	Apresentação da disciplina, Introdução ao java
2	Programação em Java, vetores, matrizes, ArrayList, arquivos
3	Modelagem de classes: introdução a classes e objetos
4	Introdução a programação Orientada a objetos, classes, métodos construtores, get e set
5	Construtores, this, sobrecarga e encapsulamento
6	Tratamento de exceções
7	Organização de pacotes, MVC
8	Composição
9	Exercícios de revisão
10	Prova I

Cronograma

11	Java2D - Possivelmente para o trabalho
12	Herança e polimorfismo: classes abstratas, inner classes
13	Interfaces, enumerações
14	Generics, set e map
15	Java 8 - Novidades
16	Java 8 - Novidades
17	Exercícios de revisão
18	Prova II
19	Apresentação de trabalhos
20	Substitutiva / Recuperação

Extra ementa

- Boas práticas de programação
- Erros de programação comuns
- Dicas de desempenho / complexidade de algoritmos
- Prevenção de erros

Avaliação

Prova I && II

Parte teórica e parte prática.

Teórica no início da prova, sem consulta e sem computador.

Após a entrega da parte teórica o aluno recebe a parte prática.

Trabalho Final

Aplicação completa

Avaliação

Nota final

$$\text{Prova I} \times 0,4 + \text{Prova II} \times 0,4 + \text{Trabalho} \times 0,2$$

Recuperação

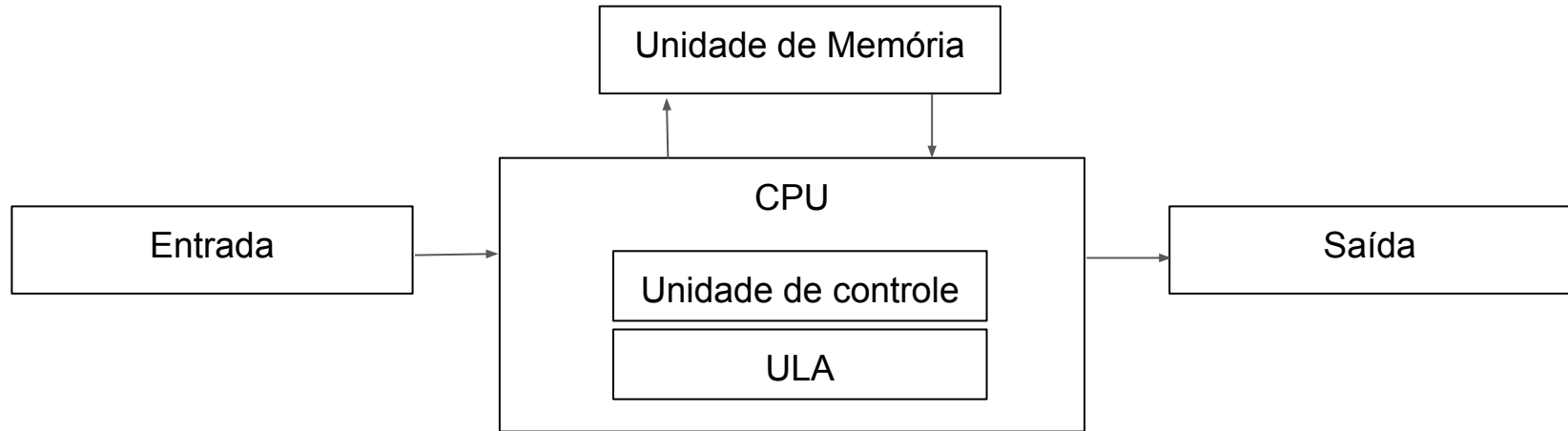
Prova final substituindo a nota mais baixa

** O trabalho não é recuperável

Welcome to JAVA

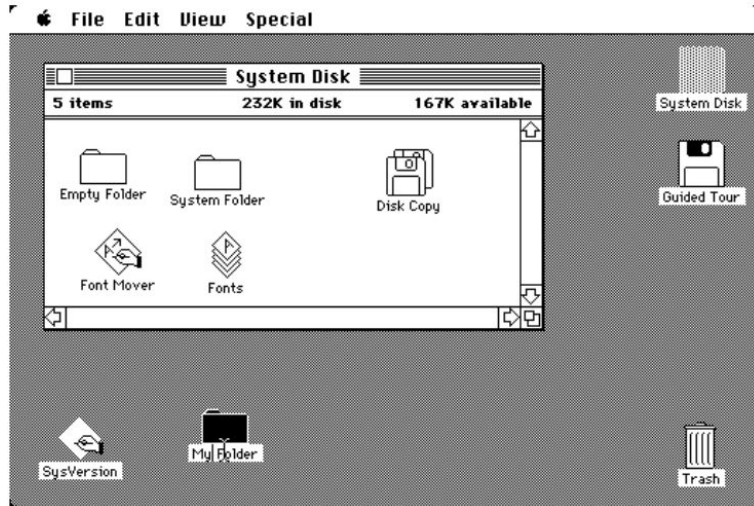
Introdução - Programação

- Hardware x Software
- Organização do Computador

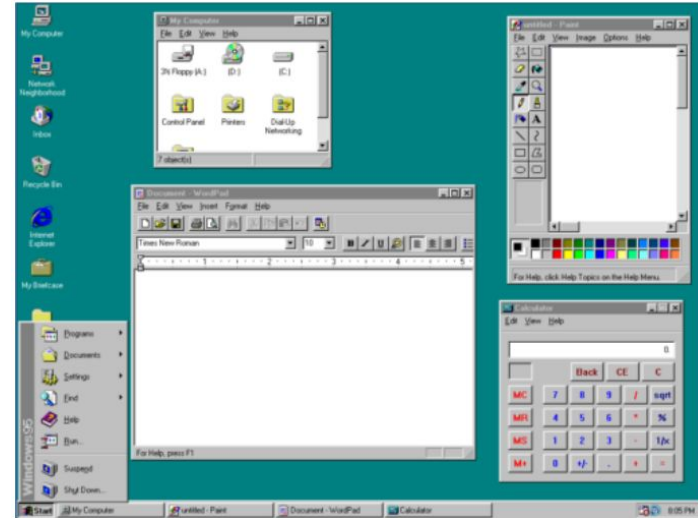


Introdução - Programação

- Primeiros Sistemas operacionais com interface gráfica



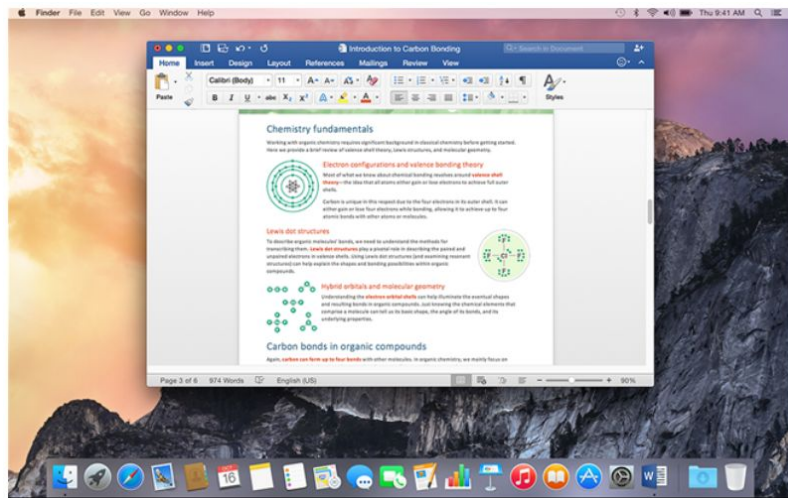
Mac OS



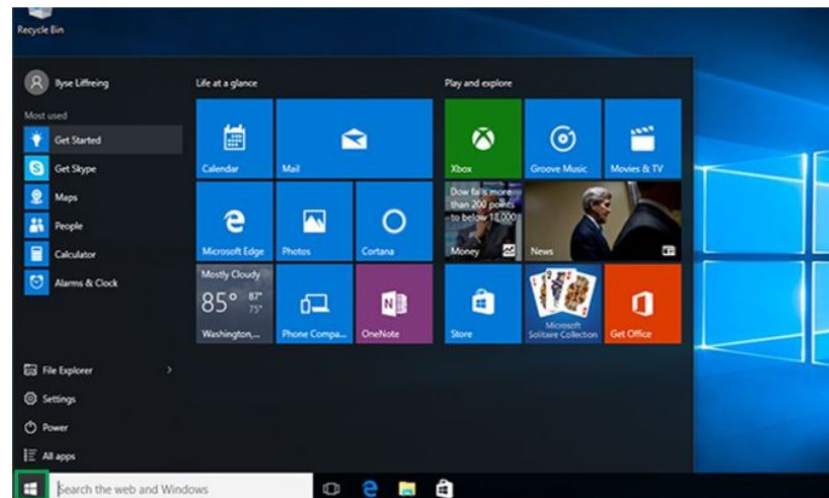
Windows 95

Introdução - Programação

- Hoje em dia



Mac OSX



Windows 10

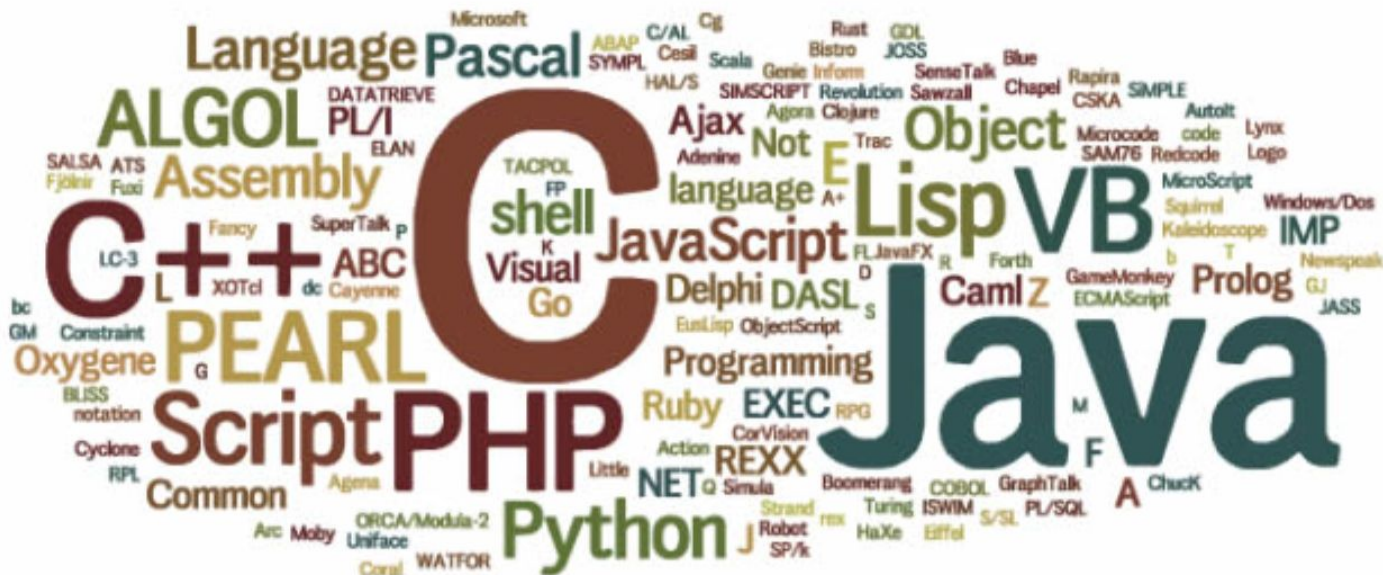
Introdução - Programação

- Internet



Introdução - Programação

- Linguagens de Programação



Introdução - Programação

História do Java

“A Sun anunciou o Java formalmente em uma conferência do setor em maio de 1995. O Java chamou a atenção da comunidade de negócios por causa do enorme interesse na Web.”

Introdução - Programação

História do Java

A empresa Oracle comprou a Sun em 2009, desde então vem mantendo os projetos relacionados ao Java e lançando novas atualizações.

Hoje o Java se encontra na versão 8 e está presente em milhares de dispositivos, sendo uma das linguagens de programação mais utilizadas no mundo.

Introdução - Programação

Por que Java?

- C Like Language
- Programe em e para qualquer plataforma.
- Comunidades ativas
- Grande número de Frameworks
- O Java não roda somente Java

Introdução - Programação

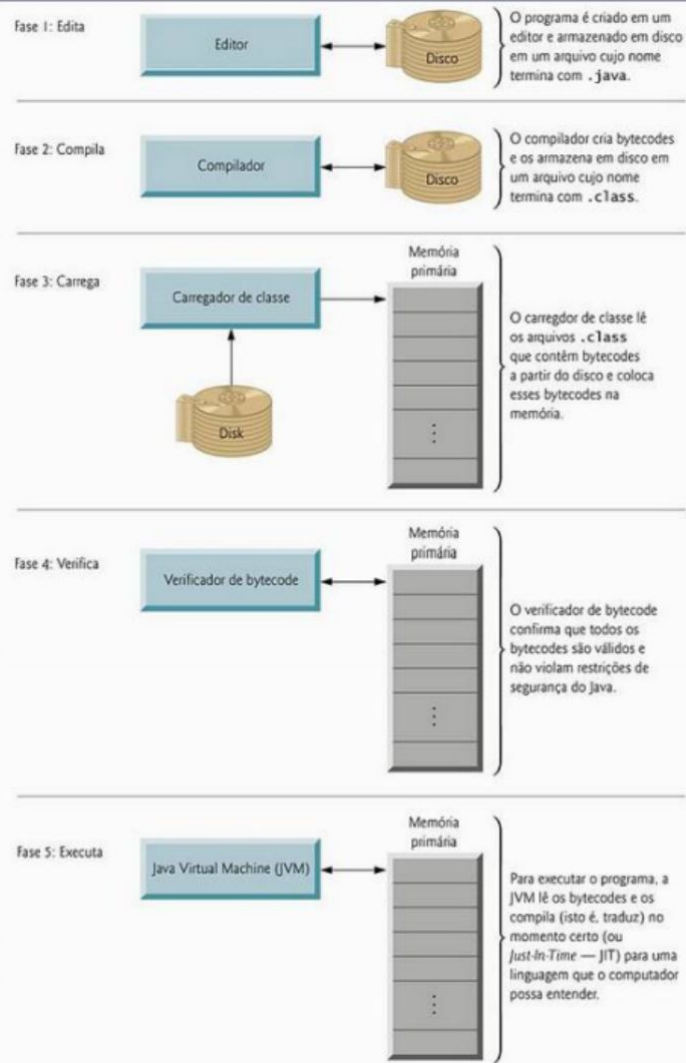
Plataformas Java

- JSE - Java Standard Edition
- JME - Java Micro Edition
- Java TV
- Java FX
- JEE - Java Enterprise Edition
- JSP, JPA, JSF ...

Introdução - Programação

Ambiente de desenvolvimento JAVA

- Fase 1 - criar o código.
`javac hello.java`
- Fase 2 - compilação
- Fase 3 - carregar na memória
- Fase 4 - verificação de bytecode
- Fase 5 - execução
`java hello`



Baby Steps

Hello World - Código fonte

```
1  //Arquivo Hello.java
2  public class Hello {
3
4      //método principal
5      public static void main(String[] args) {
6
7          System.out.print("Hello ");
8          System.out.println("World");
9          System.out.printf("%s", "!!!");
10
11     }
12
13 }
```

Hello World - Compilando


```
javac Hello.java
```


```
java Hello
```

javac - compila o código fonte e cria um arquivo do tipo class

java - procura o arquivo com o nome Hello.class e executa-o

Hello World - Análise


 COMENTÁRIOS - são ignorados pelo compilador, nunca executados

 PALAVRAS CHAVES - palavras reservadas pela linguagem, utilizadas para definirmos o que e como queremos dizer.

 System.out.print - saída padrão do sistema, podendo variar com println e printf

 Case sensitive

 O nome da classe deve ser o mesmo nome do arquivo

 Erros de sintaxe são comuns, mas o compilador é nosso BFF

System.out - Saída de dados

- `System.out.print(String text);`
 - Saída padrão.
- `System.out.println(String text);`
 - Saída padrão com quebra de linha
- `System.out.printf(String format, String[] data);`
 - Saída com dados formatados.

System.out - Saída de dados

Sequência de escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor de tela no início da próxima linha.
<code>\t</code>	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
<code>\r</code>	Retorno de carro. Posiciona o cursor da tela no início da linha atual — não avança para a próxima linha. Qualquer saída de caracteres depois do retorno de carro sobrescreve a saída de caracteres anteriormente gerados na linha atual.
<code>\\</code>	Barras invertidas. Utilizada para imprimir um caractere de barra invertida.
<code>\"</code>	Aspas duplas. Utilizada para imprimir um caractere de aspas duplas. Por exemplo, <pre>System.out.println("\"in quotes\"");</pre> exibe <code>"in quotes"</code>

Scanner - Entrada de dados

- Uma das formas mais comuns de lermos dados em Java é utilizando a classe Scanner do pacote java.util
- Para isso é necessários importarmos no cabeçalho do nosso arquivo.

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

- No nosso programa podemos declarar uma variável do tipo Scanner e informar de onde queremos ler os dados.

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

```
Scanner input = new Scanner(String text);
```

```
Scanner input = new Scanner(File source);
```

Scanner - Entrada de dados

- A partir de agora, nossa variável input possui os métodos deste objeto e pode ler alguns tipos de dados:

```
int number = input.nextInt();
```

```
String word = input.next();
```

```
String line = input.nextLine();
```

- Muito cuidado com o buffer do teclado, após lermos uma variável simples (que não seja uma linha) e pressionarmos enter para enviar o dado para o Scanner fica armazenado uma quebra de linha no buffer. Isso normalmente causa muitos problemas e dúvidas durante o início do desenvolvimento.

Operações Aritméticas

- Podemos fazer diversas operações com as variáveis em Java. Seguem alguns exemplos:

Operação Java	Operador	Expressão algébrica	Expressão Java
Adição	+	$f + 7$	<code>f + 7</code>
Subtração	-	$p - c$	<code>p - c</code>
Multiplicação	*	bm	<code>b * m</code>
Divisão	/	x/y ou $\frac{x}{y}$ ou $x \div y$	<code>x / y</code>
Resto	%	$r \bmod s$	<code>r % s</code>

Variáveis e Tipos de Dados

- boolean
- byte
- short
- int
- long
- float
- double
- char

Toda variável deve ter um tipo de dado, pois, o tipo determina que valores a variável poderá conter e que operações poderão ser realizadas com ela.

Os tipos de dados primários/primitivos são os tipos de informações mais usuais e básicas.

Variáveis e Tipos de Dados

- boolean
- byte
- short
- int
- long
- float
- double
- char

As variáveis de tipos primitivos contêm valores simples, apropriados ao tipo da variável.

Podemos classificar em três categorias:

- Lógicos
- Numéricos
- Caracteres/Alfanuméricos

Tipos de Dados

- **boolean**
- byte
- short
- int
- long
- float
- double
- char

Tipo de de dado lógico, utilizado para armazenar um único bit (0 ou 1).

Representado pelas palavras **false** ou **true**.

Ex:

boolean ligado = **false**;

Tipos de Dados

- boolean
- **byte**
- short
- int
- long
- float
- double
- char

Tipo de de dado numérico, utilizado para armazenar um 8 bits de informação.

Representado inteiros entre -128 e 127.

Ex:

byte valor = **111**;

Tipos de Dados

- boolean
- byte
- **short**
- int
- long
- float
- double
- char

Tipo de de dado numérico, utilizado para armazenar números inteiros de 16 bits, ou seja, um número entre -32768 e 32767

Ex:

short valor = **6452**;

Tipos de Dados

- boolean
- byte
- short
- **int**
- long
- float
- double
- char

Tipo de de dado numérico, utilizado para armazenar inteiros de 32 bits.

Pode representar qualquer número inteiro -2147483648 e 2147483647.

Ex:

```
int valor = -21451352;
```

Tipos de Dados

- boolean
- byte
- short
- int
- **long**
- float
- double
- char

Tipo de de dado numérico, utilizado para armazenar números grandes (64 bits de informação). Na sua declaração precisamos colocar a letra L no final do número.

Ex:

long grande = 124125325213L;

Tipos de Dados

- boolean
- byte
- short
- int
- long
- **float**
- **double**
- char

Tipo de de dado numérico,
utilizado para armazenar
números reais.

Float 32 bits Double 64 bits

Ex:

float valor = 124.23;

double valor = 124.84923;

Tipos de Dados

- boolean
- byte
- short
- int
- long
- float
- double
- **char**

Tipo de de dados de caracteres, utilizado para armazenar uma única letra

Ex:

```
char genero = 'M';
```

Tipos de Dados - String

- Strings são utilizadas para representar texto, em Java não é necessário importar nenhuma classe, as classes do pacote java.lang podem ser utilizadas normalmente sem a importação.
- Declarando uma String:

```
String text = new String();
```

```
String text = "Programando em Java";
```

- É importante entender como funcionam as Strings em Java, excesso de concatenação aumentam a frequências com que o Garbage Collector executa impactando no uso de memória. Para resolver, utilize a classe `StringBuilder`.

Constantes

- Constantes são dados que não são alterados durante a execução do programa.
- Como em Java “tudo é objeto”, dizemos que constantes são variáveis que só recebe a palavra new uma vez.
 - Obs: Objetos constantes podem ser alterados utilizando seus métodos, por exemplo um Array pode ter elementos adicionados ou removidos.
- Para declarar que algum dado é uma constante colocamos a palavra final no início de sua declaração.

```
final int MAX_VALUE = 2_147_483_647;
```


Constantes

```
final int MAX_VALUE = 2_147_483_647;
```

- Normalmente constantes tem o nome em letras maiúsculas e suas palavras são separadas por underline.
- Valores grandes também podem ser separados por underline para aumentar sua legibilidade.
- Tentar atribuir um valor a uma constante causa um erro de compilação.
- Constantes devem ser inicializadas mas não necessariamente junto com sua declaração.

Estruturas de controle

- if

```
int max(int a, int b) {  
    if (a >= b) return a;  
    return b;  
}
```

Instruções simples: Chaves
opcionais.
+ de uma instrução: chaves
obrigatórias

- else if / else

```
String imc max(int val) {  
    if (val > 30)  
        return "Obeso";  
    else if (val > 25)  
        return "Acima do peso";  
    else  
        return "Peso normal";  
}
```

Estruturas de Controle

Operador Ternário

```
int a = 5, b = 8;
```

```
String result = a == b ? "São iguais" : "Diferentes";
```



Operador	Comparação
==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor Igual
>=	Maior Igual

Estruturas de Controle

switch - case

```
switch ( x ) {  
    case 1:  
    case 2:  
        // code  
        break;  
    case 3:  
        //code  
        break;  
    default:  
        // default é opcional  
}
```

Estruturas de controle

```
while ( condição ) {  
  
    //code  
  
}
```

Executa enquanto a condição for verdadeira, primeiro testa a condição e depois executa.

```
do {  
  
    //code  
  
} while ( condição );
```

Executa enquanto a condição for verdadeira, executando o bloco pelo menos uma vez. Primeiro executa e então testa a condição.

Estruturas de controle

```
for ( int index = 0; index < 10; index += 2) {  
    //code  
}
```

quantas vezes o //code seria executado?

Estruturas de controle

```
for ( int index = 0; index < 10; index += 2) {
```

```
    //code
```

```
}
```

```
for ( inicialização ; teste condicional ; passo )
```

- inicialização: executa no início do for, apenas uma vez.
- teste condicional: caso seja verdadeiro o bloco de código executa.
- passo: a cada iteração do laço é executado.

Atenção: todos os parâmetros podem ser omitidos, no entanto, os ponto-e-vírgulas devem ser mantidos.

Estruturas de controle

FOR MELHORADO / FOREACH

```
String[ ] nomes = {"Alexandre", "Cristiana", "Guilherme", "Vinícius"};
```

```
for (String n : nomes) {
```

```
    //code
```

```
}
```

- Cada “volta” do laço de repetição a variável n assume um valor da lista.

Método main

- O início de um programa em Java se dá no método main, que tem a seguinte assinatura:

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

Métodos

- Todo método pertence a uma classe, para que sejam chamados sem a instância do objeto eles devem ser estáticos.
- A assinatura do método consiste em:

```
visibilidade retorno nome (parametros) {  
  
}
```

- visibilidade: método público, privado, protegido, abstrato.
- retorno: tipo que irá retornar, caso não retorne nenhum valor utilizar void.
- nome: a escolha do programador.
- parametros: separados por vírgula.

Passagem de parâmetros

- Costumamos dizer que em Java “tudo é ponteiro”.
- Quando passamos um objeto como parâmetro para um método, a passagem é feita por referência.
- Quando passamos tipos primitivos (int, long, char) a passagem é feita por valor.
- É comum passarmos referências de Objetos inteiros entre os parâmetros das diversas funções que criaremos.

Introdução a Programação Orientada a Objetos

Seu primeiro Objeto

Pense em qualquer objeto do mundo real, grande, pequeno, não importa.

Liste para esse objeto

- 3 características

- 2 formas de usá-lo

Exemplo

Carro

SUV

Prata

Automático

Acelera

Freia

...

POO

A ideia por trás de POO é pensar em todos elementos da construção de um programa como objetos.

Um banco, e-commerce, blog, jogo, toda aplicação, por mais simples que seja, pode ser decomposta dessa maneira.

Isso traz muitas vantagens, reutilização de código, organização, facilidade de refatoração, diminui a complexidade na hora de adicionar novas funcionalidades.

POO - Exemplo

Vamos desenvolver uma classe Calculadora que permita as quatro operações básicas (add, sub, div, mul), retornando o resultado para o usuário.

Desafio 1:

- Tente passar 0 como divisor para o método div. Como solucionar este problema?

Desafio 2:

- Utilizando uma estrutura de repetição, adiciona um método de potência que recebe dois parâmetros, o primeiro um valor double e o segundo um inteiro indicando a quantidade de potências.

Desafio 3:

- Criar um método salvar que salvar(double valor) que salva um valor na memória da calculadora.
- Adicionar um método carregar() que retorna o valor salvo, experimente utilizar este método como parâmetro de uma operação.

Seção Extra

Boas práticas de programação

- Utilize uma abordagem de bloco de construção para criar seus programas. Evite reinventar a roda - utilize partes existentes onde quer que possível. Essa reutilização de software é um benefício fundamental da programação orientada a objetos.
- Escreva seus programas de uma maneira simples e direta. Isso é às vezes chamado KIS (Keep it simple). NÃO estenda a linguagem tentando usos bizarros.
- Leia a documentação da versão do Java que você está utilizando. Consulte-a frequentemente para certificar-se de que você está ciente da rica coleção de recursos Java e de que está utilizando esses recursos corretamente.

Boas práticas de programação

- Ao programar em Java, você geralmente utilizará os seguintes blocos de construção: classes e métodos de bibliotecas de classe, classes e métodos que você mesmo cria e classes e métodos que outros criam e tornam disponíveis para você.
- Usar classes e métodos da Java API em vez de escrever suas próprias versões pode melhorar o desempenho de programa, porque eles são cuidadosamente escritos para executar de modo eficiente. Essa técnica também diminui o tempo de desenvolvimento de programa.
- É importante testar os programas em Java em todos os sistemas que você quer executá-los, para assegurar que funcionarão corretamente.

Boas práticas de programação

- Utilize linhas e espaços em branco para aprimorar a legibilidade do programa.
- Por convenção, inicie o identificador de cada nome de classe com uma letra maiúscula e inicie cada palavra subsequente do identificador com uma letra maiúscula (CamelCase)
- Utilize indentação, recue os blocos de código dentro dos pares de chave para facilitar a leitura e organização.
- Comentários são importantes, mas prefira Javadoc.

Leitura Recomendada

- Java: Como Programar. Capítulos 1, 2, 3
- <https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>
- <https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>

Referências

Livro

Java: Como programar.

Use a Cabeça! Java

Online

<https://howtodoinjava.com>

<https://docs.oracle.com/javase/8/docs/>

Contato

vinicius.machado@osorio.ifrs.edu.br

