



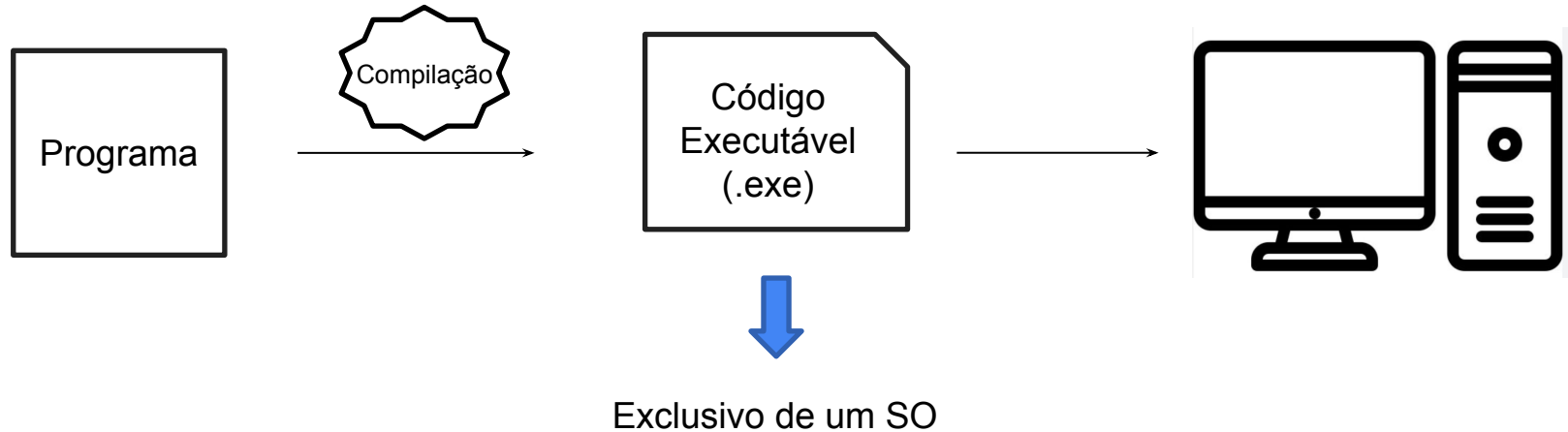
Introdução à Programação

Profº Márcio Assis
@profmarcioassis

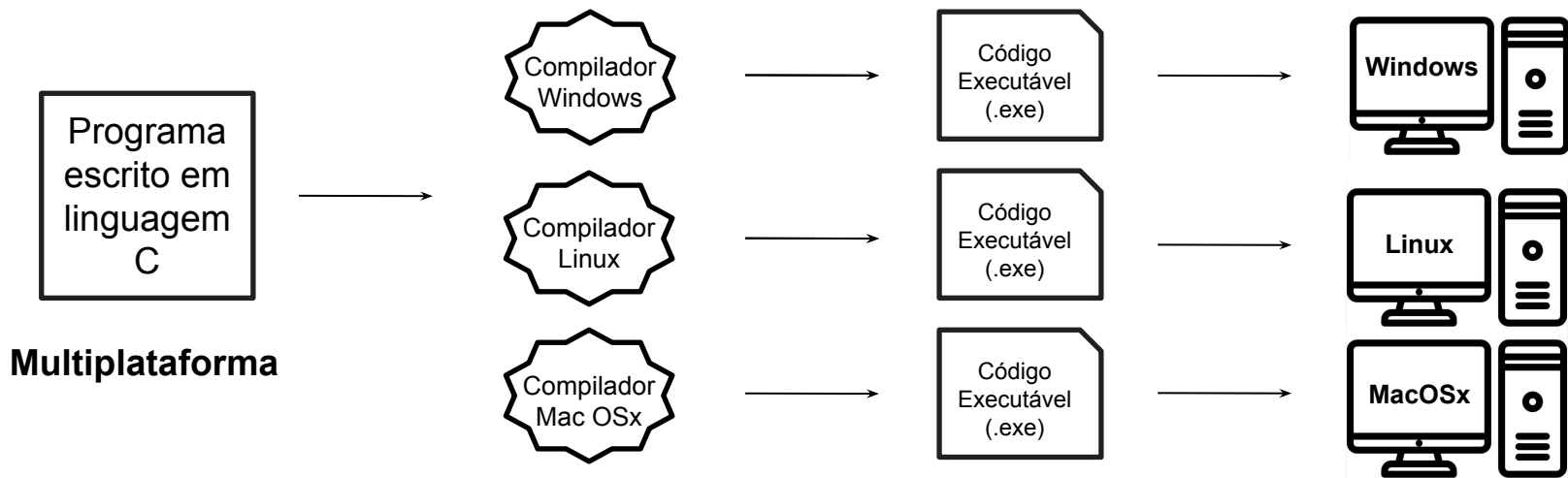
Linguagem Java

- É orientada a objetos
- É compilada para uma linguagem intermediária, chamada de **Java *bytecode***
- Os *bytecode* Java permitem que, através da **Máquina Virtual Java (JVM)**, os programas sejam **escritos uma única vez e executados em qualquer *hardware*, com qualquer sistema operacional (plataforma)**

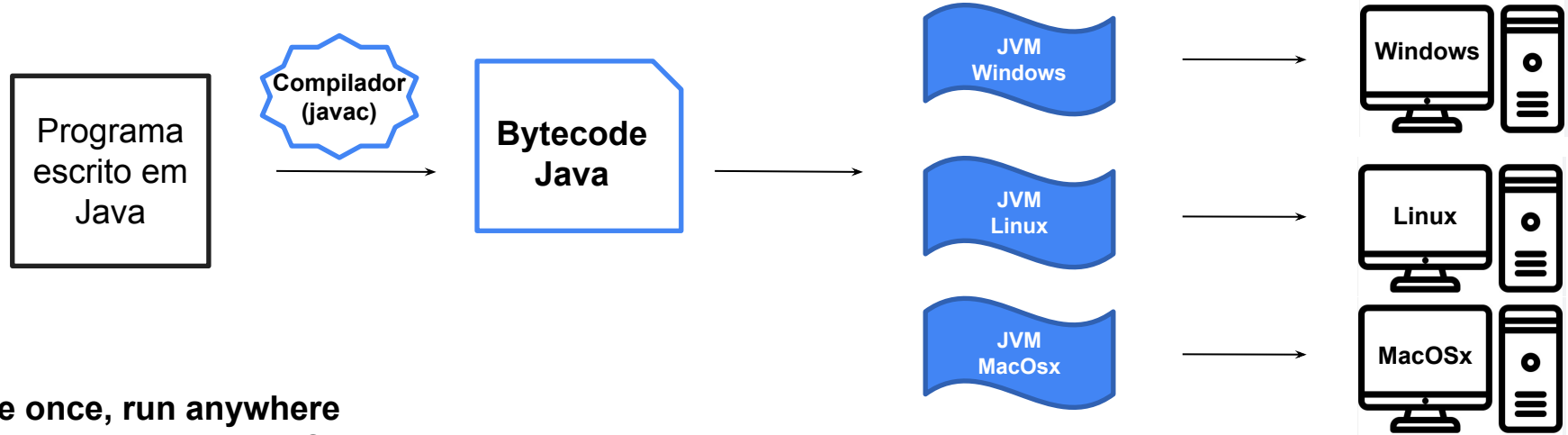
Processo de Compilação



Processo de Compilação (Linguagem C)



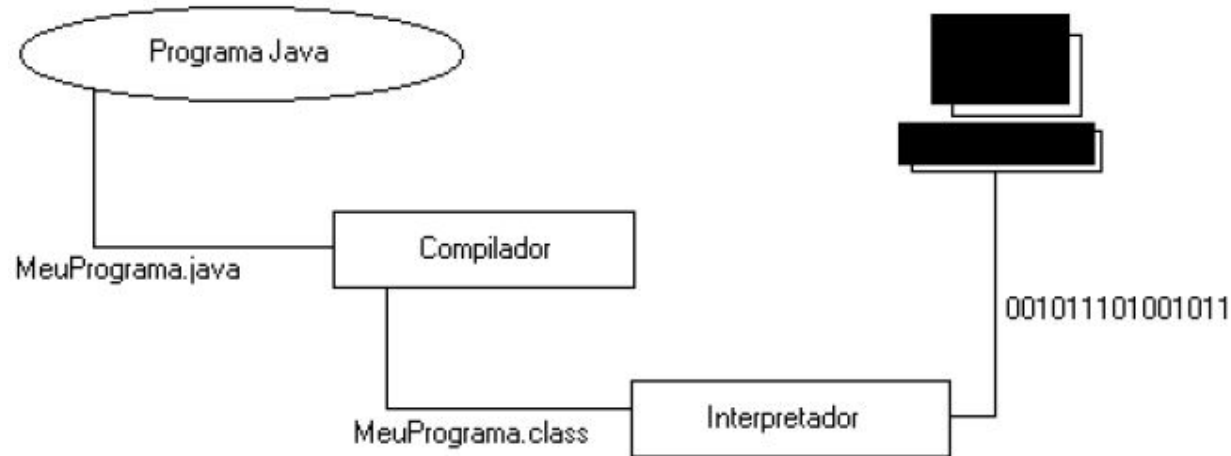
Linguagem Java



Write once, run anywhere

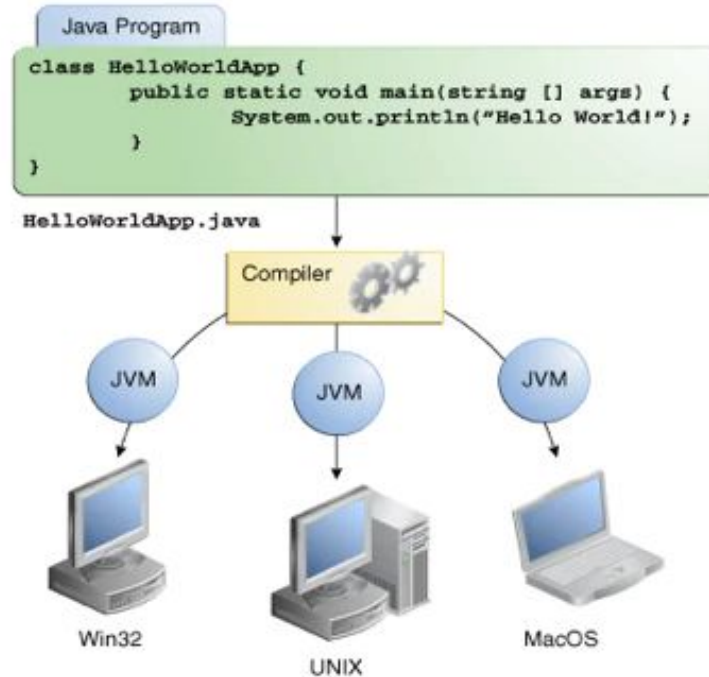
Esse é um slogan que a Sun usa para o Java, já que você não precisa reescrever partes da sua aplicação toda vez que quiser mudar de sistema operacional.

Linguagem Java



Processo de compilação e execução de um programa em Java

Plataforma Java



JVM? JRE? JDK? O que devo baixar?

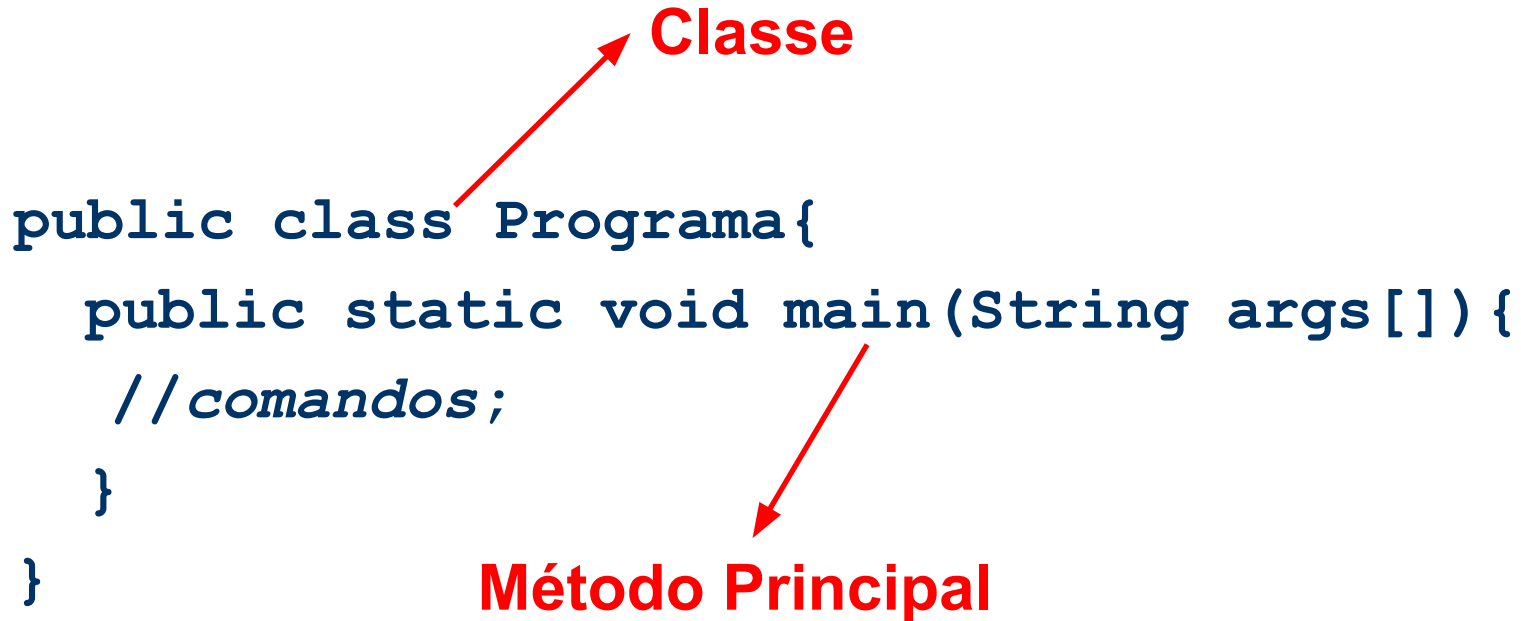
- **JVM** = apenas a virtual machine, esse download não existe, ela sempre vem acompanhada.
- **JRE = Java Runtime Environment**, ambiente de execução Java, formado pela JVM e bibliotecas, tudo que você precisa para executar uma aplicação Java.
- **JDK = Java Development Kit**: Ele é formado pela JRE somado a ferramentas, como o compilador, linguagem...
 - SE (Standard Edition)
 - EE (Enterprise Edition)
 - ME (Micro Edition)

<https://www.oracle.com/br/java/technologies/downloads/#java17>

Estrutura básica de um programa em Java

- A estrutura **básica** de um programa em Java consiste de:
 - **declaração do programa** (aqui chamado de classe);
 - sua **função principal** (aqui chamada de método), o método **main()**.

Estrutura básica de um programa em Java



The diagram illustrates the basic structure of a Java program. It features a code snippet with two red arrows pointing to specific parts. One arrow points from the word 'Classe' to the 'public class Programa{' line. Another arrow points from the text 'Método Principal' to the 'public static void main(String args[]) {' line.

```
public class Programa{  
    public static void main(String args[]) {  
        //comandos;  
    }  
}
```

Classe

Método Principal

Exemplo

```
public class PrimeiroPrograma{  
    public static void main(String args[]){  
        System.out.println("Olá Mundo!");  
    }  
}
```



Comando para exibir na tela

Como Compilar?

- Após digitar o código, salve-o como PrimeiroPrograma.java em algum diretório.
- Para compilar, você deve pedir para que o compilador (chamado javac) gerar o bytecode correspondente ao seu código Java.

```
[Suelens-MacBook-Pro:~ suelen$ cd Desktop/CodigoJava/  
[Suelens-MacBook-Pro:CodigoJava suelen$ ls  
PrimeiroPrograma.java  
[Suelens-MacBook-Pro:CodigoJava suelen$ javac PrimeiroPrograma.java  
[Suelens-MacBook-Pro:CodigoJava suelen$ ls  
PrimeiroPrograma.class PrimeiroPrograma.java  
[Suelens-MacBook-Pro:CodigoJava suelen$ █
```

Como Executar?

- Os procedimentos para executar seu programa são muito simples. O `javac` é o compilador Java, e o `java` é o responsável por invocar a máquina virtual para interpretar o seu programa.

```
PrimeiroPrograma.class PrimeiroPrograma.java
```

```
[Suelens-MacBook-Pro:CodigoJava suelen$ java PrimeiroPrograma.java
```

```
Olá Mundo!
```

```
Suelens-MacBook-Pro:CodigoJava suelen$ █
```

Declaração de variáveis

- A declaração de variáveis é realizada da seguinte forma:
- Exemplo:

```
int contador;
```

```
boolean aceito;
```

```
float salario, soma=0.0;
```

```
final float PI = 3.1415; [constante]
```

Tipos de dados

- Os tipos de dados principais encontrados em Java são os seguintes:

- Tipo Lógico

Tipo	Tamanho	Valor
<code>boolean</code>	1 bit	<code>true</code> ou <code>false</code>

- Tipo Caracter

Tipo	Tamanho	Valor
<code>char</code>	16 bits	um caractere Unicode

Tipos de dados

- ◆ Tipos inteiros

Tipo	Tamanho	Valor
byte	8 bits	-128 a 127
short	16 bits	-32768 a 32767
int	32 bits	-2147483648 a 2147483647
long	64 bits	-9223372036854775808 a 9223372036854775807

Tipos de dados

- ◆ Tipos reais

Tipo	Tamanho	Valor
<code>float</code>	32 bits	-3.40292347E+38 a +3.40292347E+38
<code>double</code>	64 bits	-1.79769313486231570E+308 a +1.79769313486231570E+308

Operadores

- ♦ Aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão (módulo)

Operadores

♦ De atribuição

Operador	Exemplo	Expressão Equivalente
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$

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

Operadores

♦ Relacionais

Operador	Significado
<code>==</code>	Igual a
<code>!=</code>	Diferente de
<code><</code>	Menor que
<code>></code>	Maior que
<code><=</code>	Menor ou igual a
<code>>=</code>	Maior ou igual a

Operadores

♦ Lógicos

Operador	Exemplo	Significado
&&	a && b	Retorna verdadeiro se a e b forem ambos verdadeiros . Senão retorna falso . Se a for falso , b não é avaliada.
&	a & b	Retorna verdadeiro se a e b forem ambos verdadeiros . Senão retorna falso . As expressões a e b são sempre avaliadas.
	a b	Retorna verdadeiro se a ou b for verdadeiro . Senão retorna falso . Se a for verdadeiro , b não é avaliada
	a b	Retorna verdadeiro se a ou b for verdadeiro . Senão retorna falso . As expressões a e b são sempre avaliadas.
!	! a	Retorna verdadeiro se a for falso . Senão retorna falso .

Operadores

```
public class Soma{  
    public static void main(String args[]){  
        double a,b;  
        a = 7;  
        b = 9.0;  
        System.out.println("a + b = " + (a + b));  
    }  
}
```

Entrada de Dados

Em Java a entrada de dados pode ser realizada via **console** ou **interface gráfica**. Inicialmente vamos utilizar a entrada de dados via console.

- Exemplo

```
import java.util.Scanner;

public class EntradaDados{
    public static void main(String args[]){
        Scanner teclado = new Scanner(System.in);
        int idade = teclado.nextInt();
        float salario = teclado.nextFloat();
        teclado.close();
    }
}
```


Entrada de Dados

Exemplo de leitura com **string** e **char**:

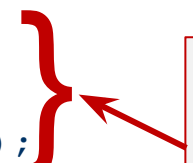
```
import java.util.Scanner;

public class EntradaDados{
    public static void main(String args[]){
        Scanner teclado = new Scanner(System.in);
        String nome = teclado.nextLine();
        char c = teclado.next().charAt(0); //para ler char
        teclado.close();
    }
}
```

Entrada de Dados (*BufferedInputStream*)

Fazendo várias leituras na mesma linha:

```
public static void main(String[] args) {  
    Scanner teclado = new Scanner(new BufferedInputStream(System.in));  
    float a = teclado.nextFloat();  
    float b = teclado.nextFloat();  
    double c = teclado.nextDouble();  
  
    System.out.printf("A = %.1f\n", a);  
    System.out.printf("B = %.2f\n", b);  
    System.out.printf("C = %.3f\n", c);  
    teclado.close();  
}
```



```
run:  
3 5 9  
A = 3,0  
B = 5,00  
C = 9,000  
BUILD SUCCESSFUL (total time: 4 seconds)  
|
```

Saída de Dados

- Semelhantemente a entrada de dados, em Java, a saída de dados pode ser realizada via **console** ou **interface gráfica**. Inicialmente vamos utilizar a saída de dados via console.

- Exemplo

```
public class SaidaDados{  
    public static void main(String args[]){  
        System.out.println("Exemplo");  
    }  
}
```

Saída formatada - *System.out.printf()*

```
import java.util.Scanner;
import java.text.DecimalFormat;
public class Exemplo {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        DecimalFormat df = new DecimalFormat("000.00");
        float A, B;
        float SOMA;
        A = teclado.nextFloat();
        B = teclado.nextFloat();
        SOMA = A + B;
        System.out.printf("SOMA = %.2f\n", SOMA);
        System.out.println("SOMA = " + df.format(SOMA));
    }
}
```

%d	representa números inteiros
%f	representa números floats
%2f	representa números doubles
%b	representa valores booleanos
%c	representa valores char

Estruturas de Controle

- Tipos de estruturas de controle
 - Estrutura de **decisão**
 - Estrutura de **repetição**

Importância das estruturas de controle

- **Determinar a execução** de instrução ou conjunto de instruções em função de uma **condição**
- Realizar a **repetição** de uma instrução ou conjunto de instruções

Tipos de estruturas de controle

- Estruturas de decisão:
 - `if...else`
 - `switch`
- Estruturas de repetição:
 - `for`
 - `while`
 - `do...while`

Onde são usadas as estruturas de controle

- Estão presentes no **interior dos métodos** declarados em classes
- Constituem o **fluxo de operações** que implementa a **lógica de negócios**

Estruturas de decisão

- **if...else**: estabelece uma **condição** para a **execução de instruções caso esta seja satisfeita** (e a execução de **outras instruções caso não seja satisfeita**)

- Estrutura:

```
if (<condição>) {  
    <comandos>;  
}  
[else {<comandos>;}]
```

Estruturas de decisão

```
public class IfElse{  
    public static void main(String args[]){  
        int idade=9;  
        if(idade < 18){  
            System.out.println("Menor de idade.");  
        }  
        else{  
            System.out.println("Maior de idade.");  
        }  
    }  
}
```

Estruturas de decisão

- Operador ? (**if ternário**): simplifica o uso de **if/else**

- Estrutura:

<atributo> = <condição> ? <expressão1> : <expressão2>

- Correspondência:

```
if (y > 0) {  
    x = 18;  
}  
else {  
    x = 36;  
}
```

Estruturas de decisão

```
public class OpCond{  
    public static void main(String args[]) {  
        int x, y;  
        y = 5;  
        x = y > 0 ? 18 : 36;  
        System.out.println("x = "+x) ;  
    }  
}
```

Estruturas de decisão

- **if...else...if**: permite implementar uma estrutura de **seleção múltipla** com **if's** aninhados (dentro de **else's**).

- Estrutura:

```
if (<condição1>) {  
    <comando1>;  
}  
else{  
    if (<condição2>) {  
        <comando2>;  
    }  
    else{  
        <comandoPadrão>;  
    }  
}
```

Estruturas de decisão

```
public class IfElseIf{
    public static void main(String args[]){
        int num = 5;
        if(num > 10){
            System.out.println("Número maior que 10");
        }
        else{
            if(num == 10){
                System.out.println("Número igual a 10");
            }
            else{
                System.out.println("Número menor que 10");
            }
        }
    }
}
```

Estruturas de decisão

- **switch**: implementa **seleção múltipla** sem a necessidade de **if...else** a partir da **avaliação de variáveis** de tipos **byte**, **short**, **char**, **int** e **String** (*a partir do Java 7*) ou uma **expressão** que as envolvam.

Estruturas de decisão

- Estrutura:

```
switch (<expressão>) {  
    case <constante1>:  
        <comandos>;  
        break; //determina a interrupção após a execução  
        ...  
    case <constanteN>:  
        <comandos>;  
        break;  
    default:  
        <comandos>;  
}
```


Estruturas de decisão

```
public class SwitchCase{
    public static void main(String args[]){
        int num = 7;
        switch(num){
            case 9:
                System.out.println("O número é igual a 9.");
                break;
            case 11:
                System.out.println("O número é igual a 11.");
                break;
            default:
                System.out.println("O número não é nem 9 e nem 11.");
        }
    }
}
```

Estruturas de repetição

- **for**: repete um conjunto de instruções, controlando o número de repetições com um **contador**.

- Estrutura:

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

Estruturas de repetição

```
public class TestFor{  
    public static void main(String args[]){  
        int cont;  
        for(cont=0; cont<=100; cont++){  
            System.out.println ("Contador é  
igual a: " + cont);  
        }  
    }  
}
```

Estruturas de repetição

- **while:** repete um conjunto de instruções **enquanto** uma certa **condição for satisfeita**.

- Estrutura:

```
while (<condição>) {  
    <comandos>;  
    <incremento>;  
}
```

Estruturas de repetição

```
public class While{  
    public static void main(String args[]){  
        int i = 0;  
        while(i < 100){  
            System.out.println("O valor de i é: " +  
i);  
            i++;  
        }  
    }  
}
```

Estruturas de repetição

- **do...while**: a condição é testada **após execução de instruções**, as quais são executadas **no mínimo uma vez**.

- Estrutura:

```
do{  
    <comandos>;  
    <incremento>;  
}while (<condição>);
```

Estruturas de repetição

```
public class DoWhile{  
    public static void main(String args[]){  
        int i = 0;  
        do{  
            System.out.println("O valor de i é: " + i);  
            i++;  
        }while( i < 100);  
    }  
}
```

Referências

- **Introdução à Programação - 2021.1**
 - Notas de aula
 - Profº Carlos Eduardo Paulino Silva
- **Programação de Computadores I - 2011.1**
 - Algoritmos: Conceito e Representação
 - Profº José Romildo Malaquias
 - Departamento de Computação – UFOP
- **Técnicas de Programação Avançada**
 - Programação Orientada a Objetos (Java): Introdução a Java
 - Profª Flávia Cristina Bernardini