Programação Orientada a Objetos

Prof. Ronald Chiesse de Souza
ronaldsouza@dcc.ufrj.br

Informações gerais do curso

- Aulas
  - Terças, de 10 às 12 horas - teoria - sala F2-004
  - Quintas, de 10 às 12 horas - prática - sala LEP1
    - Listas de exercícios para casa

- Dias sem aula
  - São Jorge - 23/04/2024 (3ª feira)
  - "Corpus Christi" - 30/05/2024 (5ª feira)

Informações gerais do curso

- Avaliações: 2 provas (P1 e P2) e um trabalho prático (TP)
  - **Datas das provas e enunciado do projeto final serão divulgados em breve**
- MP = (P1+P2)/2
  - Se MP >= 7.0 com **75% de presença** ⇒ aprovado (TP se torna opcional!)
  - Nesse caso, Média final = MP.
- Média final com TP = (P1+P2)/2*0.8 + TP*0.2
  - Se Média >= 5.0 com **75% de presença** ⇒ aprovado
  - Reprovado caso contrário
- Em caso de falta a uma das provas por motivo de saúde:
  - Aluno poderá fazer a Prova de Reposição (PR) – 2ª chamada.

Monitores

- Hugo Nascimento
  - hugons@dcc.ufrj.br
- Patrick Paiva Medeiros de Albuquerque
  - patrickpma@ic.ufrj.br
- Diego Baladez
  - diegomcb@dcc.ufrj.br
- Wemerson Silva Caxias da Costa
  - wemersonscc@dcc.ufrj.br

Comunicações da turma

- Google Classroom: **dgbrcwu**
  - Envio dos slides usados
  - Envio das listas de exercícios
  - Envio do trabalho prático

Bibliografia

- Tudose, C., Junit in Action, Third Edition, Manning Publications (2020)
- Guerra, E., Design Patterns com Java: Projeto orientado a objetos guiado por padrões, Casa do Código (2011)
- Apostila Java e Orientação a Objetos (Java e Orientação a Objetos | Alura Cursos Online)

Programação Orientada a Objetos

Unidade 1 - Introdução à linguagem Java

Prof. Ronald Chiesse de Souza
ronaldsouza@dcc.ufrj.br

Ementa

- Programação modular vs. programação orientada a objetos
- O que é Java
- Sintaxe da Linguagem Java

Relembrando programação modular...

- Programação estruturada
  - C, Pascal, Fortran, Python...

- Sequência de instruções em forma de funções, que manipulam diretamente os dados

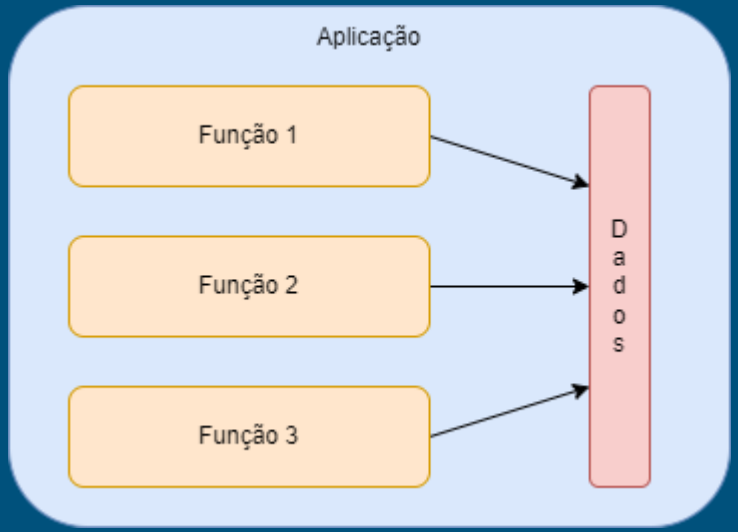
Programação orientada a objetos (POO)

- Programação não sequencial
  - Em uma mesma execução, podem existir objetos diferentes com dados diferentes
  - O conjunto de dados distribui-se entre os objetos

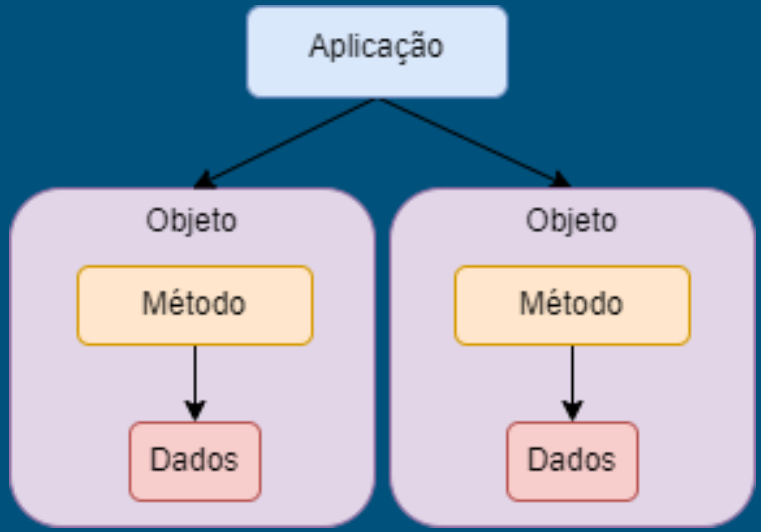
- Objetos codificam uma melhor abstração de diversos problemas do mundo real
- Funções não manipulam os dados diretamente, mas sim objetos
- Objetos podem interagir com outros objetos

Programação modular vs. POO

Programação Modular



Programação Orientada a Objetos



Linguagem de Programação Java

Como surgiu?

Máquina Virtual Java (JVM - *Java Virtual Machine*)

Versão e instalação do Java

Java

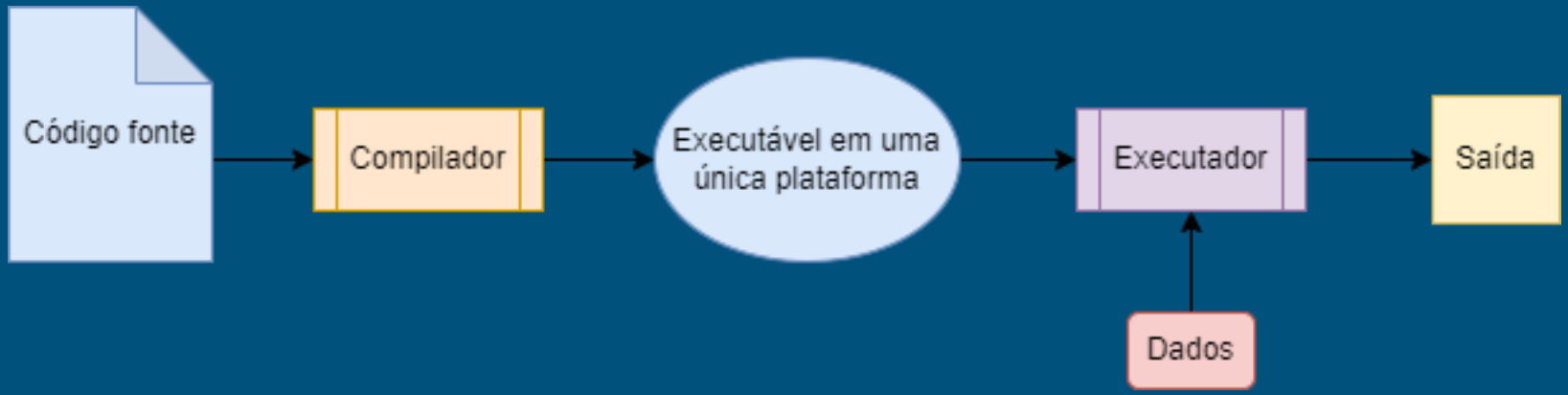
- Surgiu como uma linguagem para facilitar a programação da época
  - até então baseada em ponteiros, sem gerenciamento automático de memória; fere a legibilidade do código

- Linguagem compilada e interpretada

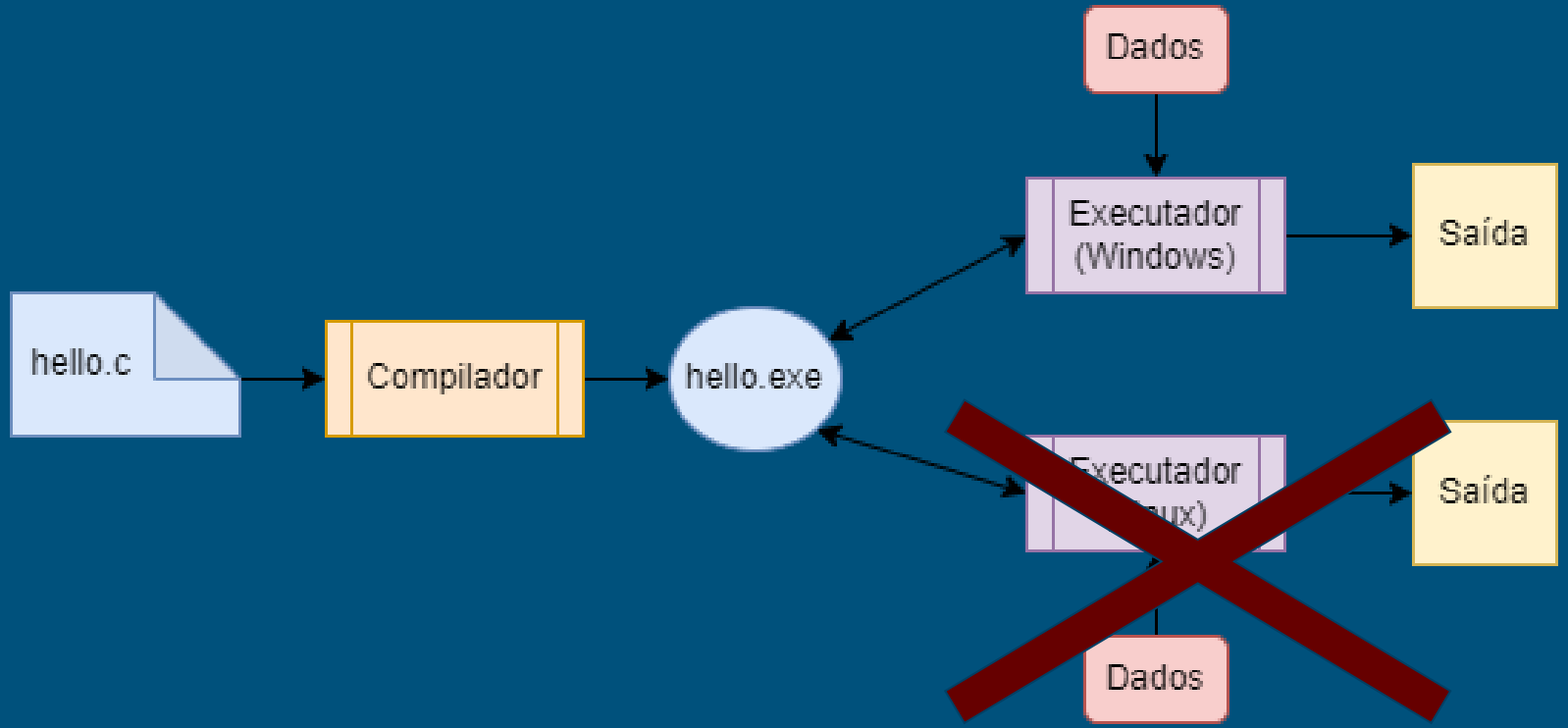
- Focada em projetos (muito) grandes e incrementais

Linguagem compilada

- Necessita de um compilador
- Cada compilação gera um executável para um SO específico

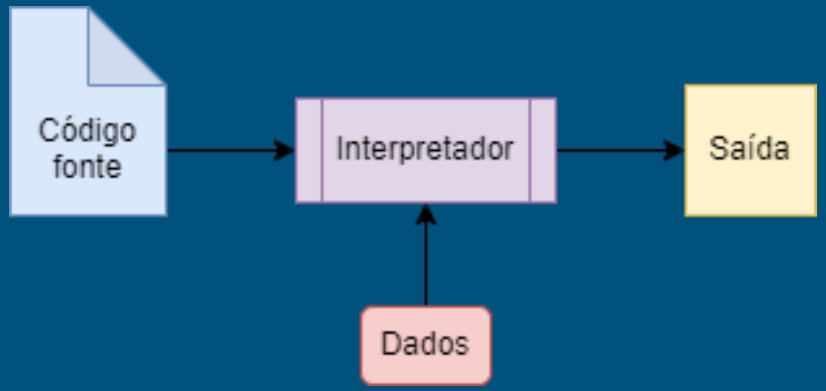


Linguagem compilada



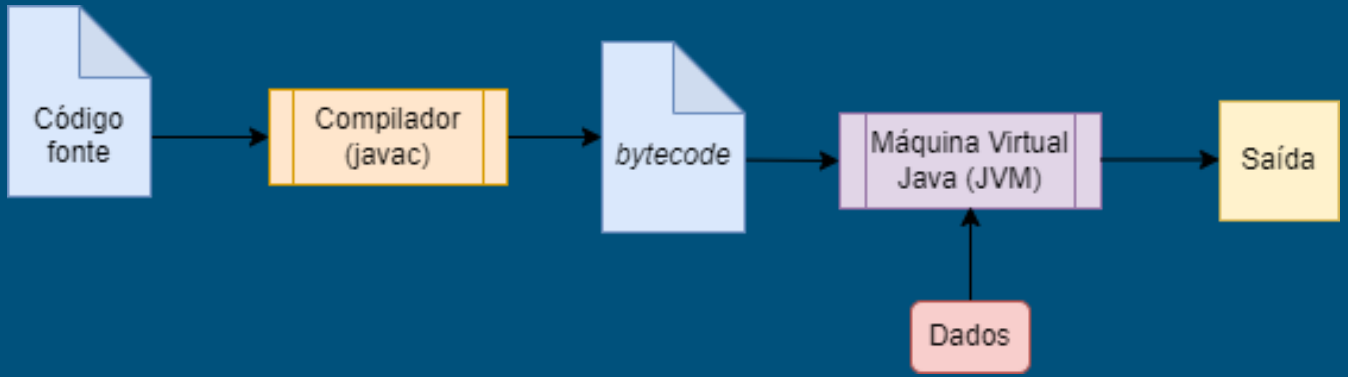
Linguagem interpretada

- Necessita de um interpretador
- Em cada execução, o interpretador lê o código fonte com os dados e retorna os resultados do código



Java: Linguagem compilada e interpretada

- Possui uma compilação parcial que transforma o código fonte em *bytecode*
- Esses *bytecodes* são passados para o interpretador do Java para serem executados
  - Interpretador do Java é uma máquina virtual para ser independente do S.O. no qual executa



Instalação do Java

- Versão que vamos utilizar no laboratório ⇒ versão 8 ou superior
  - Pode haver diferenças em determinados comandos entre versões diferentes

- Site oficial da linguagem: Java
  - JRE (*Java Runtime Environment*) → para executar aplicações em Java (bibliotecas padrão e JVM)
  - JDK (*Java Development Kit*) → para desenvolver aplicações (compilador Java e JRE)

Sintaxe da linguagem Java

Hello World em Java

Declaração de variáveis

Condicionais

Estruturas de repetição

E/S via console e teclado

Hello World em Java

```java
/* comentario de multipla linha
linha 2
*/

class PrimeiraClasse {
    public static void main (String[] args) {
        // função principal de onde o programa começa
        System.out.println("Hello World!"); // impressão na tela
    }
}
```

Declaração de variáveis

Declaração de variáveis e tipos primitivos

- Como na linguagem C, precisamos explicitar o tipo da variável que estamos declarando em Java.
- Ao atribuir uma variável A de tipo primitivo a uma outra B, o valor de A é copiado. Se A e B são de tipos distintos, tenta-se converter o valor de A para o tipo de B e não o contrário (não há conversão implícita de B para o tipo de A). Java é uma linguagem ***fortemente tipada.***

- Java é case sensitive
  - string é diferente de String

`int`	Números inteiros
`char`	Um caractere
`double`	Números decimais
`boolean`	*true* ou *false*
`String`	Sequência de caracteres

Exemplo de declaração de variáveis

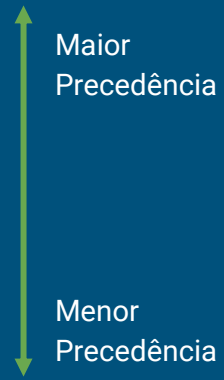
```
1  class DeclaracaoVariaveis {
2      public static void main (String[] args) {
3          String nome = "Rafaela";
4
5          int idade = 27;
6
7          double altura = 1.63;
8
9          System.out.println("Eu me chamo " + nome);
10         System.out.println("Tenho " + idade + " anos");
11         System.out.println("Minha altura é " + altura);
12     }
13 }
```

Condicionais

Operadores aritméticos e precedência

Lista de Operadores Matemáticos:

*	Multiplicação
/	Divisão
%	Resto da Divisão
+	Soma
-	Subtração



As operações matemáticas acontecem seguindo uma ordem de precedência, que é a mostrada na tabela.

Qual o resultado da seguinte operação?

```
100 + 5 * 2 - 60 / 3
```

A. 190

B. 16.667

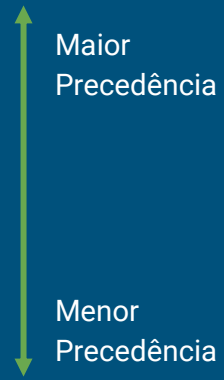
C. 90

D. 50

Operadores aritméticos e precedência

Lista de Operadores Matemáticos:

*	Multiplicação
/	Divisão
%	Resto da Divisão
+	Soma
-	Subtração



As operações matemáticas acontecem seguindo uma ordem de precedência, que é a mostrada na tabela.

Qual o resultado da seguinte operação?

```
100 + 5 * 2 - 60 / 3
```

A. 190

B. 16.667

C. 90

D. 50

Caso o Resultado desejado fosse 50, deveríamos usar parênteses para alterar a ordem de precedência. Facilitando também a leitura do código.

```
((100 + 5) * 2 - 60) / 3
```

Operadores Lógicos

Lista de Operadores Lógicos iguais a linguagem C:

==	Igualdade
!=	Desigualdade
>	Maior que
<	Menor que
>=	Maior ou igual à
<=	Menor ou igual à

&&	Operador E (AND)
||	Operador OU (OR)
!	Operador NÂO (NOT)

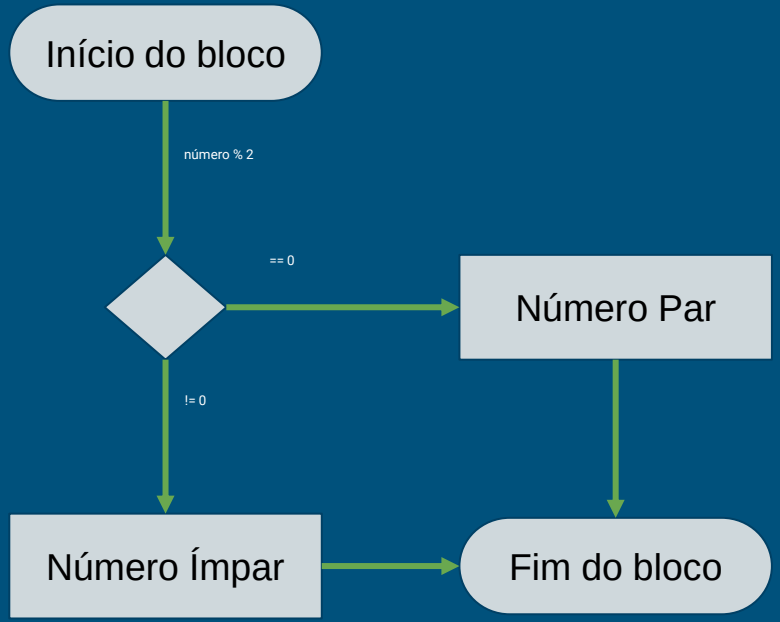
Comando *if - if else - else*

```
if(condição) {
        // comandos se condição é verdadeira
}
else if(condição 2) {
        // comandos se “condição 2” é verdadeira
}
else {
        // comandos se condições anteriores são falsas
}
```

Exemplo *if - if else - else*

Vamos criar um programa que diz se um número é par ou ímpar:

```
if(numero % 2 == 0) {
    System.out.printf(“o número é par”);
}
else {
    System.out.printf(“o número é ímpar”);
}
```

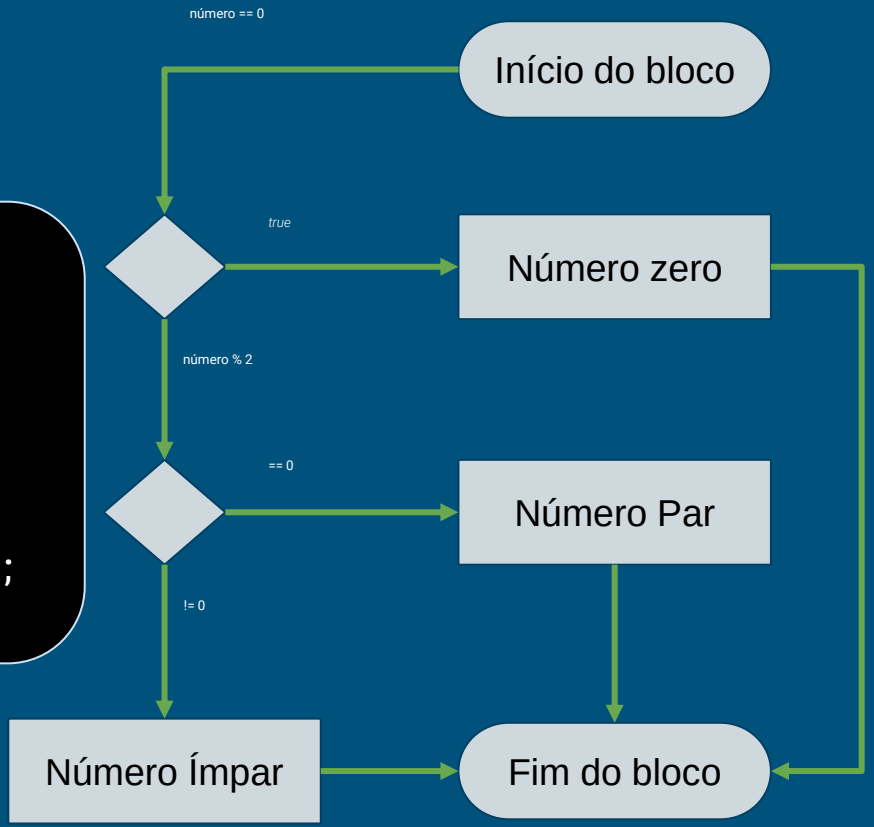


Exemplo *if - if else - else*

Vamos criar um programa que diz se um número é par ou ímpar ou zero:

```
if(numero == 0) {
    System.out.printf(“O número é zero”);
}
else if(numero % 2 == 0) {
    System.out.printf(“o número é par”);
}
else {
    System.out.printf(“o número é ímpar”);
}
```

A comparação com 0 deve ser a primeira, pois `0 % 2 == 0`.

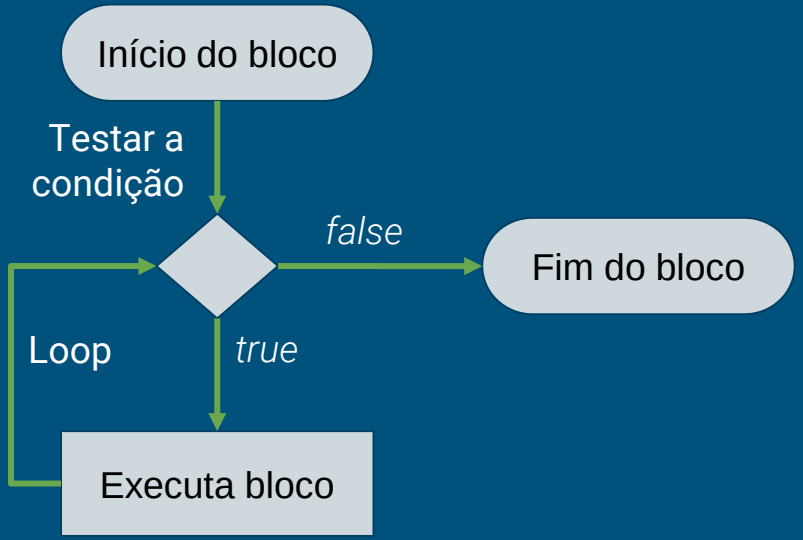


Estruturas de Repetição

While

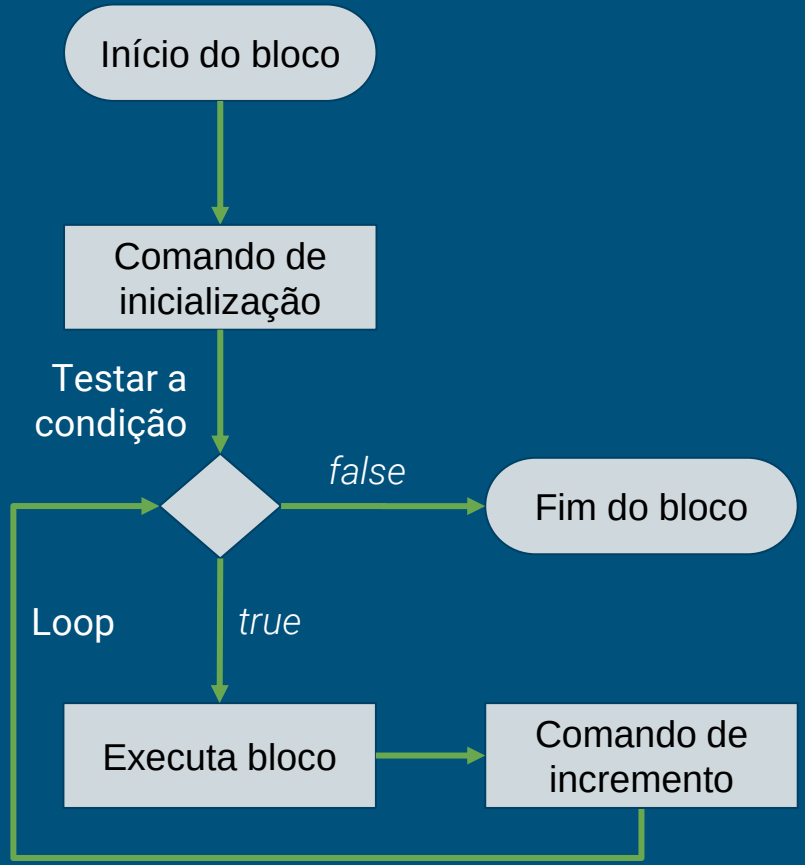
- Em um bloco de *while*, o programa testa uma condição, e caso ela seja verdadeira, entra no loop de repetição.
- O loop só será terminado quando a condição não for mais verdadeira, por isso é necessário que haja um comando que altere a condição. Caso não haja, o programa entra em um loop infinito.

```
while(condição) {
    // comandos...
}
```



For

- Na construção de bloco de *for*, além da condição de execução, podemos também colocar um comando executado no início do loop, e um que é executado ao final de cada interação.

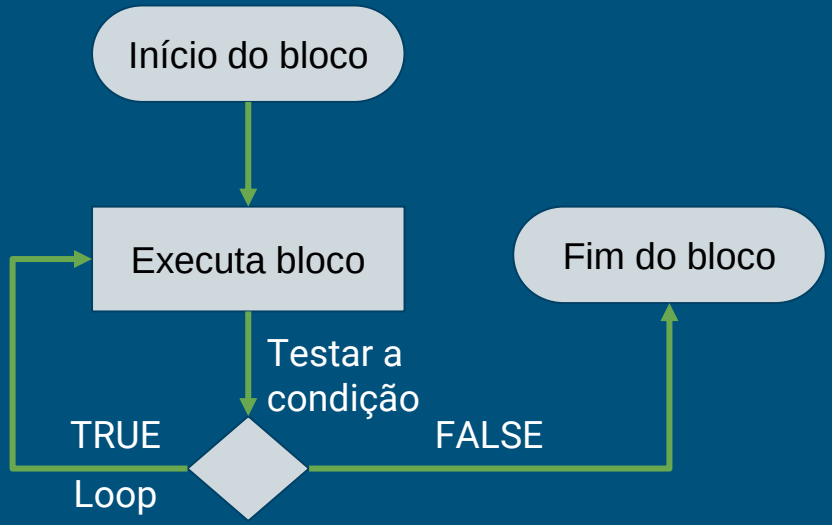



```
for(inicialização; condição; incremento) {
    // comandos...
}
```

Do-While

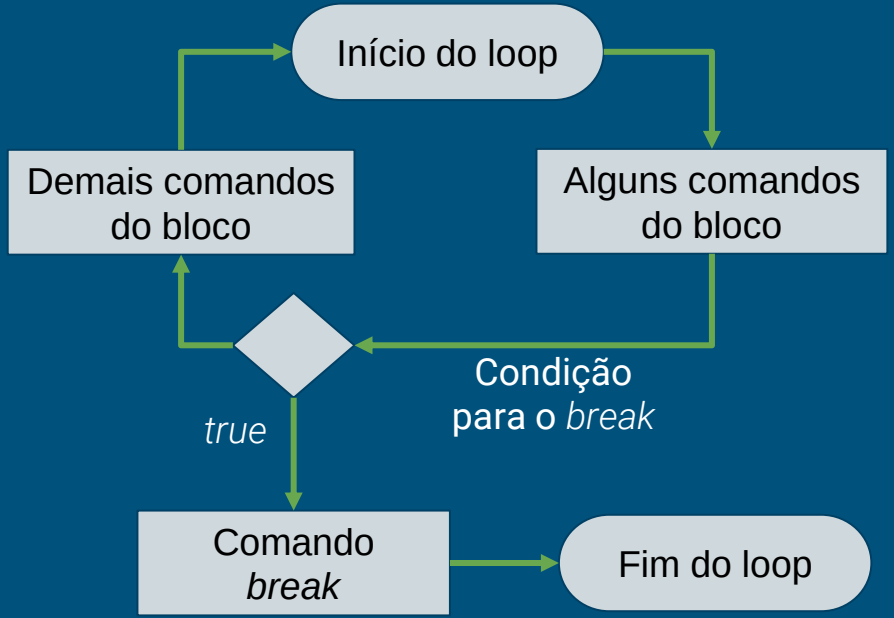
- Semelhante ao while, mas executa o bloco antes de testar a condição.
- Portanto, executa o bloco ao menos uma vez, ainda que a condição seja inicialmente falsa.

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

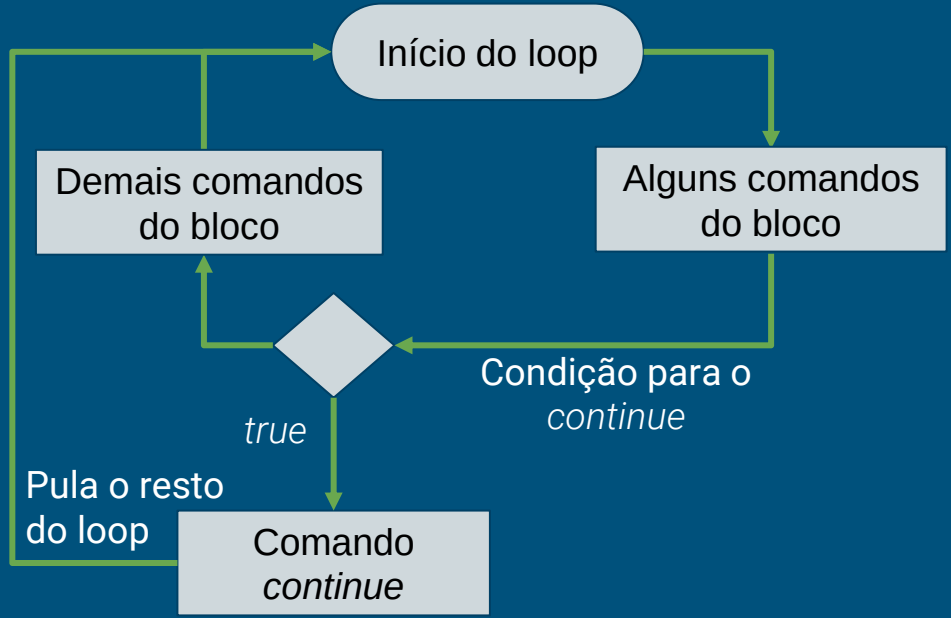


Controlando o Loop: *break* e *continue*

O desvio incondicional break permite interromper um laço antes que o bloco seja executado integralmente.



Analogamente, o desvio incondicional continue avança para a próxima iteração.



Entrada e Saída de Informações

E/S via teclado e console

- Em Java, usamos a classe System para os métodos de entrada e saída da tela
- Por exemplo, para imprimir uma String:

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

```
Olá, Mundo!
```

- Para ler informações precisamos de um Scanner

```
Scanner sc = new Scanner(System.in);
String nome = sc.nextLine();
int idade = sc.nextInt();
```

Aula prática

Esqueleto de programas Java (Hello World)

Exercícios

Atividades encontram-se descritas no pdf do Lab1 (classroom).