

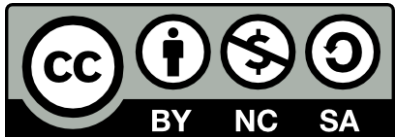
# LP3L3 – Linguagem de Programação III

2020.2



## Aula 02b

### Estruturas de Controle e Repetição



Prof. Everton Silva  
*everton.silva@ifsp.edu.br*



# Aula de Hoje

- Operadores Lógicos
- Estruturas Condicionais
- Break x Continue
- Estruturas de Repetição
- Exercícios

# Operadores Lógicos

- Todas as estruturas de controle de fluxo de um programa são baseadas em operadores lógicos:
  - `==` retorna o valor booleano `true` se os dois valores forem exatamente iguais e `false` se forem diferentes;
  - `!=` Retorna o valor booleano `true` se os dois valores forem diferentes e `false` se forem iguais;
  - `>` retorna o valor booleano `true` se o valor à esquerda for maior que o valor à direita e `false` caso for menor ou igual;
  - `<` retorna o valor booleano `true` se o valor à esquerda for menor que o valor à direita e `false` caso for maior ou igual.

# Operadores Lógicos

- Todas as estruturas de controle de fluxo de um programa são baseadas em operadores lógicos:
  - `>=` retorna o valor booleano true se o valor à esquerda for maior ou igual ao valor a direita, e false caso for menor;
  - `<=` retorna o valor booleano true se o valor à esquerda for menor ou igual ao valor a direita, e false caso for maior;
  - `&&` retorna valor booleano true se e somente se os dois valores valerem true (operado AND);
  - `||` retorna valor booleano true se ao menos um dos dois valores valerem true (operado OR).

# Estruturas Condicionais

- Normalmente a execução das instruções é realizada de maneira sequencial;
- Existem instruções que permitem especificar qual a próxima instrução a ser executada;
- Java contém três tipos de instruções condicionais;
- A instrução `if` realiza uma ação se uma condição for verdadeira ou pula a condição caso contrário.
- A instrução `if . . . else` realiza uma ação se uma condição for verdadeira ou executa uma ação diferente se a condição for falsa;
- A instrução de seleção `switch` realiza uma de muitas ações diferentes, dependendo do valor de uma expressão.

# Instrução if

- Permite que um comando ou bloco de comandos seja executado;
- Instrução em pseudocódigo:  
    Se a nota do aluno for maior que ou igual a 60  
        Imprima “Aprovado”
- Em Java  

```
if ( notaDoAluno >= 60 )  
    System.out.println("Aprovado");
```

# Instrução if...else

- Permite especificar uma ação quando a condição é true e uma ação diferente quando a condição é false;
- Instrução em pseudocódigo:
  - Se a nota do aluno for maior que ou igual a 60
  - Imprima “Aprovado”
  - Senão (else)
  - Imprima “Reprovado”
- Em Java

```
if ( notaDoAluno >= 60 )  
    System.out.println("Aprovado");  
else  
    System.out.println("Reprovado");
```

# Instrução if...else

- São ifs dentro de outros ifs. Obs: comando else sempre se refere ao if mais próximo;

- Em Java

```
if (i < 10) {  
    System.out.printf("O numero %d é menor que 10\n",i);  
}else{  
    if (i == 10) {  
        System.out.printf("O numero %d é menor igual a 10\n",i);  
    }else{  
        System.out.printf("O numero %d é maior ou igual a  
        10\n",i);  
    }  
}
```



# Instrução if...else

- São ifs dentro de outros ifs. Obs: comando else sempre se refere ao if mais próximo;

- Em Java

```
if (i < 10) {  
    System.out.printf("O numero %d é menor que 10\n",i);  
}else{  
    if (i == 10) {  
        System.out.printf("O numero %d é menor igual a 10\n",i);  
    }else{  
        System.out.printf("O numero %d é maior ou igual a  
        10\n",i);  
    }  
}
```

# Instrução switch

- Realiza ações diferentes com base nos possíveis valores de uma expressão

```
String diaDaSemana = "Sexta";  
switch (diaDaSemana) {  
    case "Domingo":  
        System.out.println("Você informou o dia 1");  
        break;  
    case "Segunda":  
        System.out.println("Você informou o dia 2");  
        break;  
    case "Terça":  
        System.out.println("Você informou o dia 3");  
        break;  
    ...  
}
```

# Break x Continue

- O Java fornece a instrução `break` para alterar o fluxo de controle;
  - A instrução `break` ocasiona a saída imediata;
  - A execução continua a com a primeira instrução depois do fluxo de controle;
- A instrução `continue` é muito utilizada em estruturas de repetição Java;
  - Sua função é ignorar o código e não sair como acontece com o `break`;

# Break

```
12 public class ExemploBreak {
13
14     public static void main(String args[]) {
15         for (int contador = 1; contador <= 1000; contador++) {
16             System.out.println("Esta é a repetição nr: " + contador);
17             if (contador == 10) {
18                 break;
19             }
20         }
21     }
22 }
```

# Continue

```
12 public class ExemploContinue {
13
14     public static void main(String args[]) {
15         for (int contador = 1; contador <= 100; contador++) {
16             if (contador % 5 != 0) {
17                 continue;
18             }
19             /* Se o contador não for múltiplo de 5
20              * Todo o código abaixo será ignorado
21              * e o loop continua com o próximo nr */
22             System.out.println("Contador: " + contador);
23         }
24     }
25 }
```

# Estruturas de Repetição

- Utilizada para repetir um comando ou grupo de comandos diversas vezes consecutivas;
- Ex.: fatorial de um número;
- Diferentes linguagens de programação dispõem de diferentes tipos de estruturas de repetição;
- Cada tipo de estrutura de repetição possui uma característica específica, indicada para lidar com problemas específicos;
- As estruturas de repetição mais conhecidas são o `while`, `for` e `do-while`.

# Estruturas de Repetição

- Característica comum a todas as estruturas de repetição de qualquer linguagem de programação:
  - o número de repetições sempre precisa ser finito, independente se o mesmo é, ou não, conhecido;
- Contadores são variáveis que recebem um valor inicial e são modificadas a cada iteração.

# while

- Forma básica:

```
while (condição)  
    comando_ou_bloco;
```

- Exemplo:

```
int product = 3;  
while (product <= 100)  
    product = 3 * 100;  
System.out.println("Valor de product: " + product);
```



# do-while

- Forma básica:

```
do-while{comando_ou_bloco}  
    while(condicao);
```

- Exemplo:

```
Scanner s = new Scanner(System.in);  
String nome;  
do{  
    System.out.println("Digite seu nome:");  
    nome = s.nextLine();  
}while(nome.isEmpty());  
System.out.println("Olá, " + nome);
```

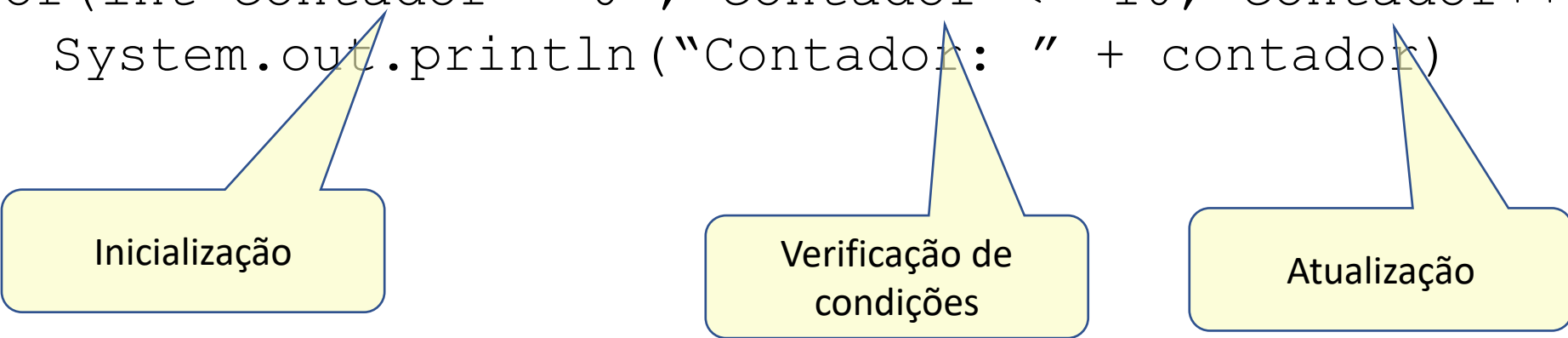
# for

- Forma básica:

```
for(inicialização; verificação_de_condições; atualização)  
    comando_ou_bloco;
```

- Exemplo:

```
for(int contador = 0 ; contador <= 10; contador++ )  
    System.out.println("Contador: " + contador)
```



Inicialização

Verificação de  
condições

Atualização

# Exercícios

1. Colocar estas duas linhas iniciais no método main():

```
Random rnd = new Random(); //Inicia Aleatório  
int x = rnd.nextInt(100); //Gera um número aleatório (0 – 99)
```

- Faça um laço de repetição que solicite ao usuário digitar um número;
  - O laço encerra quando o usuário acertar o número;
  - Se o número do usuário for menor que o oculto, escrever: “MAIOR”, se o número do usuário for maior que o oculto, escrever: “MENOR”.
2. Leia um número positivo do usuário, então calcule e imprima a sequencia Fibonacci até o primeiro número superior ao número lido. Exemplo: se o usuário informou o número 30, a sequencia a ser impressa será 0 1 1 2 3 5 8 13 21 34.

# Dúvidas?

