Aula 3

- Operadores aritméticos unários *(Livro, pág. 132-133)*
- Instrução de atribuição com operação
- Estruturas de controlo – repetição *(Livro, pág. 181-192)*
- Instrução repetitiva `for`
- Instrução repetitiva `while` e `do...while`
- Instruções de salto `break` e `continue`

Operadores aritméticos unários

- incremento de 1: `++ (++x, x++)`
- decremento de 1: `-- (--x, x--)`
- Os operadores de incremento e decremento atualizam o valor de uma variável com mais ou menos uma unidade.
- Colocados antes são pré-incremento e pré-decremento. Neste caso a variável é primeiro alterada antes de ser usada.

```
Y = ++X; // equivalente a: x = x + 1; y = x;
```

- Colocados depois são pós-incremento e pós-decremento e neste caso a variável é primeiro usada na expressão onde está inserida e depois atualizada.

```
Y = X++; // equivalente a: y = x; x = x + 1;
```

Atribuição com operação

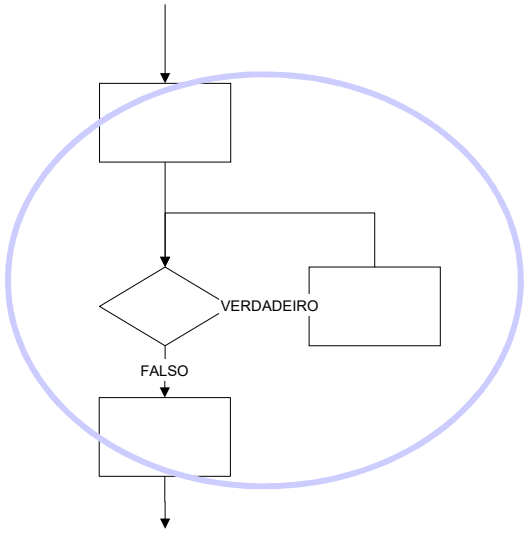
- É comum usar uma versão compacta do operador de atribuição (=) onde este é precedido de uma operação (por exemplo `+=, -= *=, /=, %=,...`).
- A instrução resultante é equivalente a uma instrução normal de atribuição em que a mesma variável aparece em ambos os lados do operador =.
- A importância desta notação tem a ver com a simplificação do código e com a clareza da operação a realizar.

```
int x, y, z;
...
y += 5;          // equivalente a y = y + 5;
z *= 5 + x;      // equivalente a z = z * (5 + x);
y += ++x;        // x = x + 1; y = y + x;
```

Estruturas de controlo - repetição

- Para além da execução condicional de instruções, por vezes existe a necessidade de executar instruções repetidamente.
- A um conjunto de instruções que são executadas repetidamente designamos por ciclo.
- Um ciclo é constituído por uma estrutura de controlo que controla quantas vezes as instruções vão ser repetidas.
- As estruturas de controlo podem ser do tipo contador (`for`) ou do tipo condicional (`while` e `do…while`).
- Normalmente utilizamos as estruturas do tipo condicional quando o número de iterações é desconhecido e as estruturas do tipo contador quando sabemos à partida o número de iterações.

Diagramas de Fluxo – *Flowcharts* (Ciclo for)



Enquanto for verdadeiro FAZ..

Testa no início

Ciclo *for*

```
for(inicialização ; condição ; atualização)
{
    instruções;
}
```

- A `inicialização` é executada em primeiro lugar e apenas uma vez.
- A `condição` é avaliada no início de todos os ciclos e as instruções são executadas enquanto a `condição` for verdadeira.
- A parte da `atualização` é feita no final de todas as iterações.
- Em geral, a função da `inicialização` e da `atualização` é manipular variáveis de contagem utilizadas dentro do ciclo.

Exemplo - Tabuada

- Impressão da tabuada de n com n <= 10:

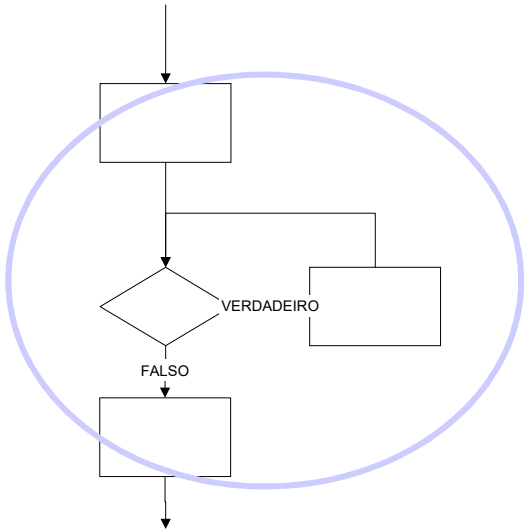
```
// Repetindo as instruções várias vezes…pouco eficiente
int  n=5;
System.out.printf("%2d X %2d = %3d\n", n, 0, n*0);
System.out.printf("%2d X %2d = %3d\n", n, 1, n*1);
System.out.printf("%2d X %2d = %3d\n", n, 2, n*2);
…
System.out.printf("%2d X %2d = %3d\n", n, 10, n*10);
```

```
// Controlando a repetição com um ciclo for
for(i = 0 ; i <= 10 ; i++){
    System.out.printf("%2d X %2d = %3d\n", n, i, n*i);
}
```

Exemplo – Factorial de n!

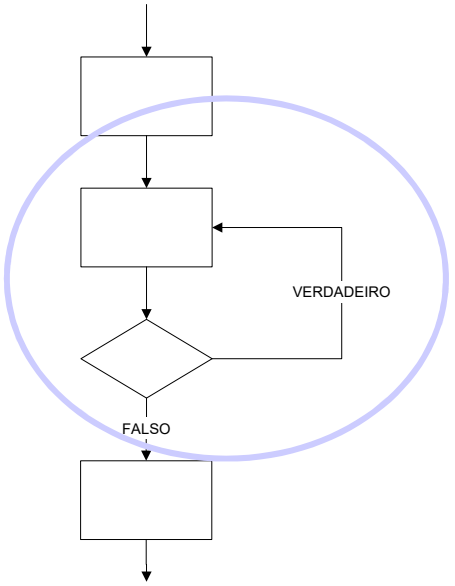
```
// Factorial de n! = 1 x 2 x 3 x …. x (n-1) x n

Scanner ler = new Scanner(System.in);
int n, fact;
System.out.print("Factorial de: ");
n = ler.nextInt();
fact =1;
for (int i = 1; i<= n; i++) {
    fact = fact * i;
}
System.out.printf(" %3d! = % d %n", n, fact);
// Experimentar com vários valores de n. Criticar resultados!
```

Diagramas de Fluxo – *Flowcharts* (Ciclos *while*)



Enquanto for verdadeiro FAZ..

Testa no início (for, while)



FAZ... Enquanto for verdadeiro

Testa no fim (do...while)

Ciclos *While, do...while*

```
do
{
   instruções;
}while(condição);
```

```
while(condição)
{
   instruções;
}
```

- A sequência de instruções colocadas no corpo do ciclo são executadas enquanto a condição for verdadeira.
- Quando a condição for falsa, o ciclo termina e o programa continua a executar o que se seguir.
- A diferença principal entre as duas instruções repetitivas reside no facto de no ciclo `do … while` a sequência de instruções é executada pelo menos uma vez.
- Muito cuidado na definição da condição!
  - Pode nunca entrar ou nunca sair do ciclo!

Exemplos - leitura de um valor inteiro positivo

```
int x, cont = 0;
do{
    System.out.print("Um valor inteiro positivo: ");
    x = sc.nextInt();
    cont++;
} while (x <= 0);
System.out.printf("Valor %d lido na %d vez\n",x,cont);
```

```
int x = -1, cont = 0; // Atenção à inicialização de x
while(x <= 0){
    System.out.print("Um valor inteiro positivo: ");
    x = sc.nextInt();
    cont++;}
System.out.printf("Valor %d lido na %d vez\n",x,cont);
```

Exemplo – Cálculo da média de uma sequência

```
// Lê uma sequência de números positivos e calcula a média
Scanner ler = new Scanner(System.in);
int nota, soma=0, num=0;
float media;
System.out.println("Introduza as notas dos alunos (<0 termina):");
nota = ler.nextInt();
while (nota >= 0) {
        soma = soma + nota;
        num = num +1;
         nota = ler.nextInt();
}
media = (float) soma / num;
System.out.printf("Soma = %3d\nMedia = %4.1f\n", soma, media);
```

Break e continue

- Podemos terminar a execução de um bloco de instruções com duas instruções especiais: `break` e `continue`.
- A instrução `break` permite a saída imediata do bloco de código que está a ser executado. É usada normalmente no `switch` e pode ser usada em estruturas de repetição, terminando-as.
- A instrução `continue` pode ser usada para terminar a execução do bloco de instruções dentro de um ciclo, forçando a passagem para a iteração seguinte (não termina o ciclo).

Exemplo - *break*

```
int x, cont = 0;
do{
    System.out.print("Um valor inteiro positivo: ");
    x = sc.nextInt();
    cont++;
    if(cont >= 10) break; //depois de 10 tentativas, termina o ciclo
} while(x <= 0);
if(x > 0){
    System.out.printf("Valor %d lido em %d
        tentativas\n",x,cont);
} else {
    System.out.printf("Ultrapassadas 10 tentativas\n");
}
```

Exemplo - *continue*

```
int i, n, soma = 0;
do{
    System.out.print("Valor de N [1 ... 99]: ");
    n = sc.nextInt();
} while(n < 1 || n > 100);

for(i = 1 ; i <= n ; i++){
    // se numero par avança para a iteração seguinte
    if(i % 2 == 0)continue;
    soma += i;
}
System.out.printf("A soma dos impares é %d\n", soma);
```

break e continue - **USAR com adequação**

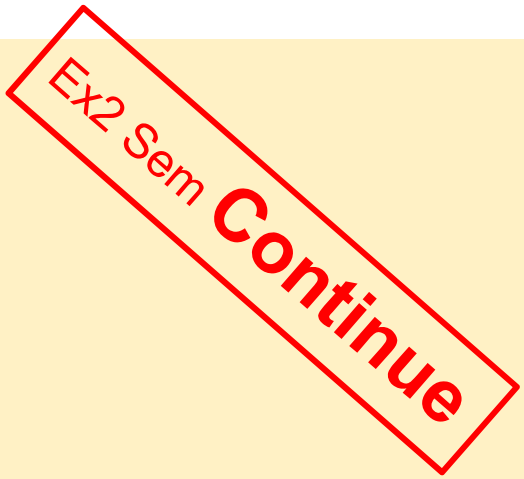
Sempre que o **uso de break/continue dificulte a compreensão** dos programas e algoritmos, pela introdução de vários pontos de saída**, deve ser evitado o seu uso**. A dificuldade aumenta com a dimensão dos programas. Para programas pequenos podem ser adequados.

Podem ser substituídos por construções ***if*** e/ou ***condições de teste*** adequadas.

```
int x, cont = 0;
do {
        System.out.print("Um valor inteiro positivo:");
        x = ler.nextInt();
        cont++;
} while (x <= 0 && cont < 3); // termina ao fim de 3 tentativas
if (x > 0) {
        System.out.printf("Valor %d lido em %d tentativas\n", x, cont);
} else {
        System.out.printf("Ultrapassadas 3 tentativas\n");
}
```

Ex1 Sem **Break**

break e continue (2)




```
int i, n, soma = 0;
do {
    System.out.print("Valor de N [1 ... 99]: ");
    n = ler.nextInt();
} while (n < 1 || n > 100);
for (i = 1; i <= n; i++) { // outra alternativa for (i=1; i<=n; i=i+2){soma += I;}
    if (i % 2 != 0) { // se número for par avança para a iteração seguinte
        soma += i;
    }
}
System.out.printf("A soma dos impares é %d\n", soma);
```