



Aula 4

Ciclos, Vectores

Iniciativa Conjunta:



Com o apoio de:





Porquê utilizar-se Ciclos?

- O Java permite a utilização de um tipo de estruturas (ciclos) que permitem repetir uma operação ou uma sequência de operações sucessivas.

```
Enquanto i < 100 faz:  
    System.out.println("Aula 4");
```

- Quando se constrói um ciclo define-se uma guarda cujo valor dá continuidade ou interrompe o ciclo.

```
i < 100
```

- Sendo um conceito fundamental para a programação em Java são definidos três tipos formatos:
 - Ciclo **while**;
 - Ciclo **do-while**;
 - Ciclo **for**.

While

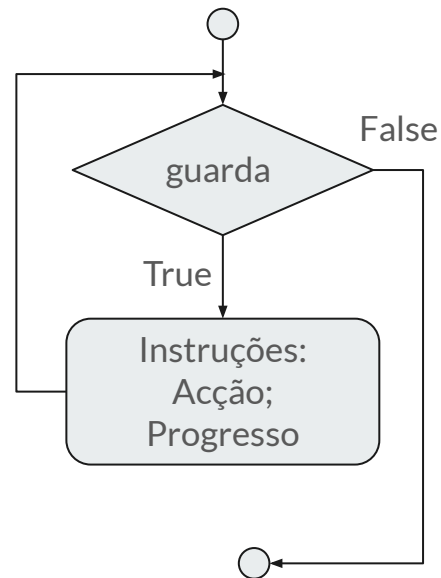
- Sintaxe:

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

- No exemplo anterior:

```
int i = 0;  
while ( i < 100 ) {  
    System.out.println("Aula 4");  
    i = i + 1;  
}
```

Importante para obrigar a guarda a passar a false!





While

- Qual o resultado esperado do seguinte código.

```
int soma = 0;
int i = 0;
while ( i < 10 ) {
    soma = soma +i;
}
System.out.println("O valor é:" + soma);
```

- Seria um ciclo infinito pois a variável *i* é sempre <10.



While

- A guarda é sempre verdadeira porque a variável **i** é sempre **<10**, devia-se ter incrementado a variável **i**.

```
int soma = 0;
int i = 0;
while ( i < 10 ) {
    soma = soma +i;
    i++;
}
System.out.println("O valor é:" + soma);
```

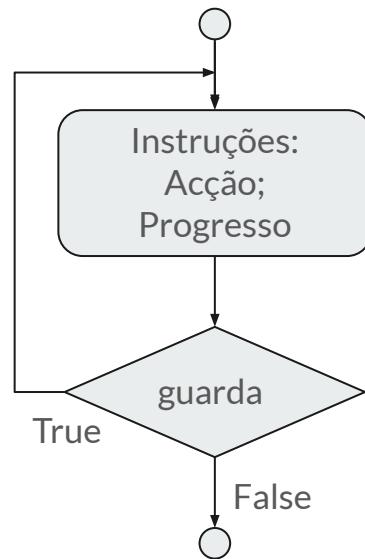
Do-while

- Variação da estrutura do ciclo **while**. Sintaxe:

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

- No exemplo inicial:

```
int i = 0;  
do {  
    System.out.println("Aula 4");  
    i++;  
} while ( i < 100 );
```



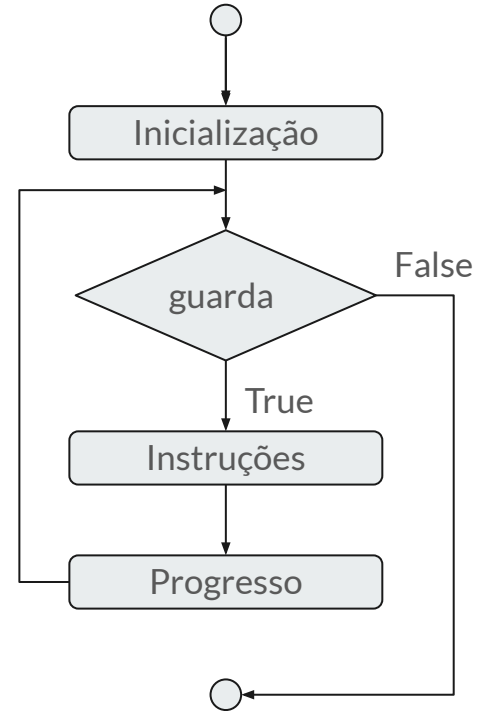
For

- Sintaxe:

```
for ( inicialização; guarda; progresso; ) {  
    Instruções;  
}
```

- No exemplo inicial:

```
for ( int i = 0; i < 100; i++ ) {  
    System.out.println("Aula 4");  
}
```





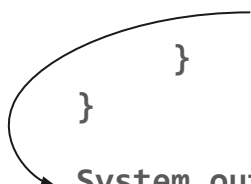
Que ciclo usar?

- Os três tipos de ciclos apresentados são equivalentes nas suas funções.
- Os ciclos **for** e **while** são *pretest loops* pois a condição da guarda é validada antes das instruções serem executadas. O ciclo **do-while** é *posttest loop* pois valida a guarda após as instruções serem executadas.
- Por norma o ciclo **for** pode ser utilizado quando se conhece, no início, o número de repetições pretendidas. Já o ciclo **while** pode ser utilizado quando não se sabe quantas repetições vão ser realizadas.

Break, é possível usar em ciclos?

- À semelhança das estruturas **switch** também nos ciclos é possível utilizar a expressão **break**. A utilização desta expressão termina no imediato o ciclo.

```
int soma = 0;
int i = 0;
while ( i < 20 ) {
    soma = soma + i;
    i++;
    if ( soma >= 100 ) {
        break;
    }
}
System.out.println("O valor é:" + soma);
```





Exercício 1

- Escreva os números pares de 1 a 20, usando as três estruturas de ciclos apresentadas.

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

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

```
for ( inicialização; guarda; progresso; ) {  
    Instruções;  
}
```



Recursividade

- Quando uma função contém **invocações a si própria**, esta é considerada uma função recursiva.
- Recursividade é um conceito fundamental em computação. Nalguns paradigmas de programação (por exemplo, no paradigma funcional), a recursividade assume uma importância fulcral.
- Dada a sua proximidade com as definições matemáticas, a “elegância” das definições de funções recursivas tornam-as atrativas em certos contextos.



Exemplo - Sucessão de Fibonacci

- Definição **matemática** da função para obter o n-ésimo número da sucessão de Fibonacci

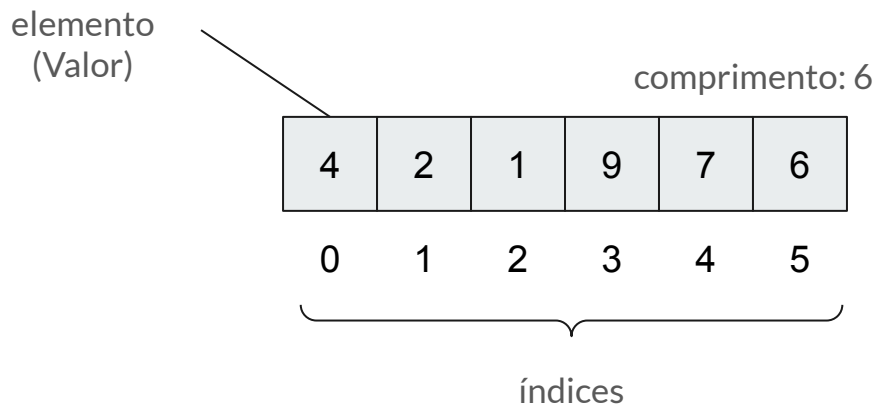
$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & \text{outros casos} \end{cases}$$

- Função recursiva **em Java**

```
static int fibonacci(int n) {  
    if(n <= 1) {  
        return n;  
    } else {  
        return fibonacci(n - 1) + fibonacci(n - 2);  
    }  
}
```

Vectores

- Estrutura elementar que permite armazenar dados de forma organizada.
- Os acessos aos elementos do vetor são efectuados mediante um índice.



Criação de vectores

- Em Java, os vectores têm comprimento fixo. Desta forma, ao ser criado um vector é necessário indicar um comprimento, o qual não pode ser alterado após a criação.

```
int[] v1 = new int[3];  
boolean[] v2 = new boolean[2];
```

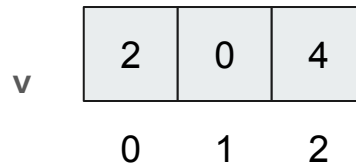
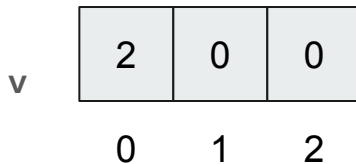
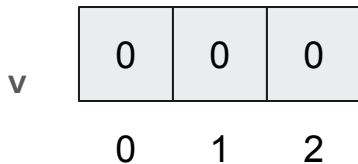
- Quando os vectores são criados, os elementos tomam um valor por omissão. No caso de um vector de inteiros esse valor é **0**, já no caso de um vector de *boolean* este valor é *false*.



Manipulação de vetores: Modificação

- A modificação dos elementos do vector é feita mediante um índice.

```
int[] v = new int[3];  
v[0] = 2;  
v[2] = 4;
```



Manipulação de vectores: Acesso

- O comprimento de um vector pode ser obtido através do atributo *length*.

```
int size = v.length;  
int first = v[0];  
int last = v[v.length-1];
```

```
System.out.println("O tamanho do vector é: " + size);  
System.out.println("O primeiro elemento do vector é: " + first);  
System.out.println("O último elemento do vector é: " + last);
```

Output:

```
O tamanho do vector é: 3  
O primeiro elemento do vector é: 2  
O último elemento do vector é: 4
```

v	2	0	4
	0	1	2

Manipulação de vectores: Iteração

- O processo de percorrer os elementos de um vector pode ser designado por iteração. Tipicamente, a iteração faz uso de uma variável que toma sucessivamente os valores dos índices do vector que se quer aceder (variável *i*).

```
static int sum(int[] v) {  
    int i = 0;  
    int sum = 0;  
    while(i != v.length) {  
        sum = sum + v[i];  
        i = i + 1;  
    }  
    return sum;  
}
```

v	2	0	4
	0	1	2

Manipulação de vetores: Iteração

- O que faz a função que se segue?

```
static int sum(int[] v) {  
    int i = 0;  
    int sum = 0;  
    while(i != v.length) {  
        sum = sum + v[i];  
        i = i + 1;  
    }  
    return sum;  
}
```

v

2	0	4
0	1	2

Manipulação de vectores: Iteração

- O que faz a função que se segue?

```
static int sum(int[] v) {  
    int i = 0;  
    int sum = 0;  
    while(i != v.length) {  
        sum = sum + v[i];  
        i = i + 1;  
    }  
    return sum;  
}
```

v

2	0	4
0	1	2

- Calcula o somatório de todos os elementos do vector na variável **sum**.

Manipulação de vetores: Iteração

- Ao iterar-se sobre um vetor é também possível contar-se o número de ocorrência de determinado valor no vetor

```
static int numberOfOccurrences(int a, int[] v) {  
    int i = 0;  
    int count = 0;  
    while(i != v.length) {  
        if ( v[i] == a ){  
            count = count + 1;  
        }  
        i = i + 1;  
    }  
    return count;  
}
```

v

2	0	4
0	1	2



Manipulação de vectores: Iteração

- É possível também pesquisar num vector a existência de um determinado valor. Qual seria o código que possibilitaria esta acção?
- E o código que devolveria o valor máximo de um vector, como seria?