



# CIÊNCIAS DA COMPUTAÇÃO

FACULDADE DE CIÊNCIAS | UNIVERSIDADE AGOSTINHO NETO

1º Ano

## FUNDAMENTOS DE PROGRAMAÇÃO

Docente:

✓ Mateus Padoca Calado - PhD

Monitores:

✓ Nsimba Kiafuka  
✓ Mariano Calelua



## Tema - 03

- Estruturas de Repetição
- Arrays(vectores e matrizes)
- ArrayList
- String



## Cap. IV

## Estruturas de Repetição

# CAP. IV - Estruturas de Repetição

- ❑ As estruturas de repetição são usadas para executar de forma repetida um conjunto de instruções.
- ❑ Usam condição que define o número de vezes que as instruções serão executadas.
- ❑ Em Java as estruturas de repetição são:
  - `for()`
  - `while()`
  - `do while();`

# CAP. IV - Estruturas de Repetição – for()

- ❑ A estrutura **for** é utilizada na maioria das vezes quando sabemos o número de vezes que se deseja repetir as instruções.

SINTAXE - JAVA	Pseudocódigo
<pre>for(inicialização; condição; incremento){     &lt;blocos de instruções&gt; }</pre>	<pre>para V de vi ate vf passo p faça     &lt;blocos de instruções&gt; fimpara</pre>

- **Inicialização** – expressão de atribuição sobre a variável de controlo que define o valor inicial do ciclo.
- **Condição** – expressão relacional que define a número de iteração do ciclo.
- **Incremento** – define como a variavel de controlo do ciclo varia em cada iteração do ciclo.
- As três secções anterior são separadas por pontos-e-vírgulas.

# CAP. IV - Estruturas de Repetição – for()

- ❑ EX: Programa em Java que imprime a todos números inteiros de 1 à 50.

Resolução - Pseudocódigo	Resolução - JAVA
<pre>inicio     Para v de 1 ate 50 passo 1 faca         escreva("\n Numero: ", v)     fimpara fimalgoritmo</pre>	<pre>public class Fundamentos {      public static void main(String[] args) {          for (int i = 1; i &lt;= 50; i++) {             System.out.println("Numero: " + i);         }     } }</pre>

# CAP. IV - Estruturas de Repetição – while()

- ❑ **While** é uma estrutura de repetição com verificação no início. Pode ser utilizada quando não sabemos o número de vezes que as instruções serão repetidas, desde que a condição do while seja verdadeira.

SINTAXE - JAVA	Pseudocódigo
<pre>&lt;inicialização&gt; while(condição){     &lt;blocos de instruções&gt;     &lt;incremento&gt; }</pre>	<pre>&lt;inicialização&gt; enquanto (condição) faca     &lt;bloco de instruções&gt;     &lt;incremento&gt; fimenquanto</pre>

- **Inicialização** – expressão de atribuição sobre a variável de controlo que define o valor inicial do ciclo.
- **Condição** – expressão relacional que define a número de iteração do ciclo.
- **Incremento** – define como a variavel de controlo do ciclo varia em cada iteração do ciclo.

# CAP. IV - Estruturas de Repetição – while()

- ❑ EX: Programa em Java que imprime a todos números inteiros de 1 à 50.

Resolução - Pseudocódigo	Resolução - JAVA
<pre>inteiro i inicio      i &lt;- 1     enquanto (i &lt;= 50 ) faca         escreva("\n Numero: ", i)         i &lt;- i + 1     fimenquanto  Fimalgoritmo</pre>	<pre>public class Fundamentos {      public static void main(String[] args) {          int i = 1; //inicialização         while(i &lt;= 50) { //condição              System.out.println("Numero: " + i);              i++; //incremento         }     } }</pre>



# CAP. IV - Estruturas de Repetição – do..while()

- ❑ **do...while** é uma estrutura de repetição com verificação feita no fim. Ela é utilizada maioritariamente na solução de problemas que executam as instruções pelo menos uma vez. Ex: Criação de um MENU de opções

SINTAXE - JAVA	Pseudocódigo
<pre>&lt;inicialização&gt; do {     &lt;blocos de instruções&gt;     &lt;incremento&gt; }while( condição );</pre>	<pre>&lt;inicialização&gt; faca     &lt;blocos de instruções&gt;     &lt;incremento&gt; enquanto(condição)</pre>

- **Inicialização** – expressão de atribuição sobre a variável de controlo que define o valor inicial do ciclo.
- **Condição** – expressão relacional que define a número de iteração do ciclo.
- **Incremento** – define como a variavel de controlo do ciclo varia em cada iteração do ciclo.

# CAP. IV - Estruturas de Repetição – do..while()

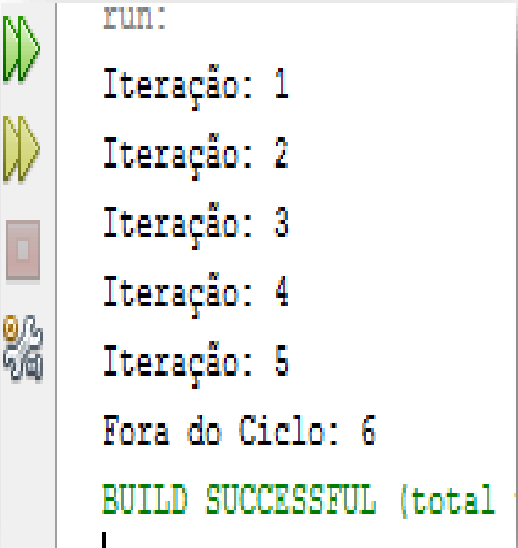
- ❑ EX: Programa em Java que imprime a todos números inteiros de 1 à 50.

Resolução - Pseudocódigo	Resolução - JAVA
<pre>inicio     i &lt;- 1     faça         escreva("\n Numero: ", i)         i &lt;- i + 1     enquanto (i &lt;= 50 ) Fimalgoritmo</pre>	<pre>public class Fundamentos {     public static void main(String[] args) {         int i = 1; //inicialização         do{             System.out.println("Numero: " + i);             i++; //incremento         }while(i&lt;50); //condição     } }</pre>

# CAP. IV - Estruturas de Repetição

## ❑ Comando Break

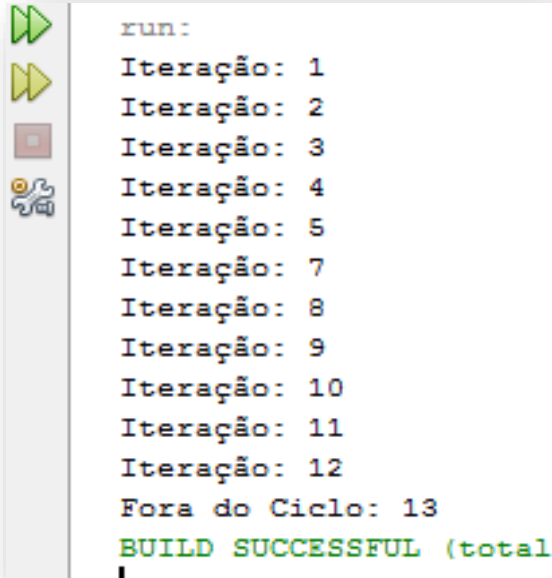
- O comando break utiliza-se nas estrutura de repetição e na estrutura switch para ocasionar uma saída imediata dessas estrutura. A execução continua na primeira instrução que vem apos a estrutura de repetição.

Exemplo	Resultado
<pre>public class Fundamentos {      public static void main(String[] args) {         int i;         for (i = 1; i &lt;= 12; i++) {             if (i == 6)                 break;              System.out.println("Iteração: " + i);         }         System.out.println("Fora do Ciclo: " + i);     } }</pre>	

# CAP. IV - Estruturas de Repetição

## ❑ Comando Continue

- O comando break utiliza-se nas estrutura de repetição para ocasionar um salto nas instruções restantes no corpo do ciclo e prosseguir para a próxima iteração do ciclo.

Exemplo	Resultado
<pre>public class Fundamentos {      public static void main(String[] args) {         int i;         for (i = 1; i &lt;= 12; i++) {             if (i == 6)                 continue;              System.out.println("Iteração: " + i);         }         System.out.println("Fora do Ciclo: " + i);     } }</pre>	 <pre>run: Iteração: 1 Iteração: 2 Iteração: 3 Iteração: 4 Iteração: 5 Iteração: 7 Iteração: 8 Iteração: 9 Iteração: 10 Iteração: 11 Iteração: 12 Fora do Ciclo: 13 BUILD SUCCESSFUL (total</pre>

# CAP. IV - Estruturas de Repetição – Exercícios

1. Crie um programa em Java que recebe uma quantidade infinita de números inteiros. O programa deverá imprimir a quantidade de números lidos menores que 8. A leitura termina quando for digitado um número Negativo.
2. Crie um programa em Java que recebe dois (2) números inteiros e imprime todos os números no intervalo do primeiro e o segundo. Considere o segundo maior do que o primeiro.
3. Crie um programa em Java que imprime a média aritmética de vários valores positivos múltiplos de 2, lidos a partir do teclado. O programa termina de ler caso seja inserido um valor negativo.

# Conteúdo



Cap. V

Arrays(vetores e matrizes)

# CAP. V - Arrays

## ❑ Motivação

- Número de variáveis que declaramos reflecte de certa forma a quantidade de elementos que desejamos manipular. Imagine que deseja armazenar três notas de um estudante?
- Neste caso o mais comum é declarar uma variável do tipo String para o Nome e três variáveis do tipo Float para as notas. Agora imagine o que seria para armazenar dados de 50 estudantes cada um com três notas? Declararias 150 variáveis?
- Ou ainda como resolveria a leitura de 20 números inteiros e que se deseja obter o maior e o menor destes? Declararias 20 variáveis?
- Claro que não 150 ou 20 variáveis normais. Mas sim vectores ou matrizes.

# CAP. V - Arrays

## ❑ Definição

- Array é uma colecção de variáveis do mesmo tipo, acessíveis com um único nome e armazenadas de forma contínua na memória.
- Em Java os índices de um vector com  $n$  elementos variam sempre entre 0 e  $n - 1$ .

$v1 =$

0	1	2	3
Lukau	Paulino	Valdano	Solange

Representação de um Array de String com 4 elementos.

$v2 =$

0	1	2	3
20	13	9	27

Representação de um Array de Inteiro com 4 elementos.



# CAP. V - Arrays

## ❑ Conceitos em Java

- **Arrays** em Java são objectos, portanto são considerados tipos por referência.
- Os elementos de um **Array** podem ser de tipos primitivos ou tipos por referencia.
- Para referenciar a um elemento do **Array** é necessário colocar o nome da variável seguido de um ou mais índices entre colchetes [ ].
- O índice de um Array deve ser um valor inteiro não negativo.

## ❑ Arrays podem ter uma ou muitas dimensões:

- Vector = 1 dimensão
- Matriz = mais do que uma dimensão

# CAP. V. Arrays - Declaração

- ❑ Em Java, um vector é declarado de várias formas, obedecendo a seguinte sintaxe: `tipo_dado [ ] nome_vector = new tipo_dado [dimensão];`

- `tipo_dado` - o tipo de dado do vector (tipo primitivo ou tipo por referencia).
- `nome_vector` - o nome da variavel Array.
- `Dimensão` - o tamanho do vector definido em valor inteiro positivo.

❑ Ex: `int [ ] num = new int [5];`

- Implica que a variável `num` tem a capacidade para armazenar 5 elementos; a sua primeira posição é 0 e a última é 4.
- Com esta opção, todas as posições de `num` são inicializadas com Zero (`0`) tendo em conta o seu tipo (`int`).

- ❑ Pode ser igualmente declarado e inicializado da seguinte maneira:

```
int [ ] num;  
num = new int [5];
```

# CAP. V. Arrays - Declaração

- ❑ **Inicialização explícita:** a inicialização é feita atribuindo valores de forma explícita ao vector utilizando { }.

❑ Ex1: `int num[ ] = {2,6,8,7,5};`

- O vector **num** possui a dimensão 5.
- Na sua primeira posição `num[0]` encontramos o valor 2, `num[1]=6`, `num[2]=8`, `num[3]=7` , `num[4]=5`.

❑ Ex2: `double[ ] m = {};`

- Para o exemplo 2, não existe alocação de espaço; isto implica que não podemos aceder a nenhuma posição do vector.

# CAP. V. Arrays - Manipulação

## ❑ Atribuição Direita

```
String nomes[]=new String [2];  
nomes[0]="Eliana";  
nomes[1]="Francisco";
```

## ❑ Leitura de dados:

```
int numeros = new int [5];  
for (int i=0;i< numeros.length;i++){  
    numeros[i]= teclado.nextInt();  
}
```

## ❑ Escrita:

```
for (int i=0; i < numeros.length; i++){  
    System.out.println(numeros[i]);  
}
```

OU:

```
for (int numero : numeros ){  
    System.out.println(numero);  
}
```

- O método `length` retorna a dimensão do vector. (Ex: `num.length` )

# CAP. V. Arrays - Manipulação

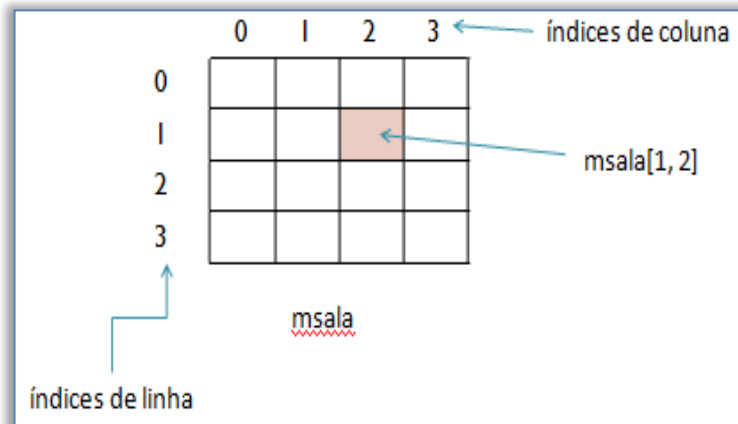
- ❑ EX: Programa em Java que lê os 10 primeiros números digitados pelo utilizador e mostra na tela.

## Resolução

```
public class Fundamentos {  
    public static void main(String[] args) {  
  
        Scanner teclado = new Scanner(System.in);  
        final int MAX = 10;  
        float numeros[] = new float[MAX];  
  
        /*Ler o números */  
        for (int i = 0; i < numeros.length; i++) {  
            System.out.print("\nDigite o número: ");  
            numeros[i] = teclado.nextFloat();  
        }  
  
        /*Mostrar os números armazenados no vector */  
        for (int i = 0; i < numeros.length; i++) {  
            System.out.print("\nPosição " + i + ": " + numeros[i]);  
        }  
    }  
}
```

# CAP. V. Arrays - Multidimensional

- ❑ Os Arrays multidimensional também denominado de matriz comportam mais do que uma dimensão organizada na forma linhas e colunas.



## ❑ Sintaxe

```
tipo_dado [][] nome_matriz = new tipo_dado [L][C];
```

- **tipo\_dado** - o tipo de dado do array (tipo primitivo ou tipo por referencia).
- **nome\_matriz** - o nome da variavel Array.
- **L**- a quantidade de linhas da matriz.
- **C**- a quantidade de colunas da matriz.

# CAP. V. Arrays - Multidimensional

## ❑ Atribuição Direita

```
int num [][] = new int[2][2];  
num[0][0] = 23;  
num[0][1] = 33;
```

## ❑ Leitura de dados:

```
int num[][] = new int[2][2];  
for (int linha = 0; linha < num.length; linha++) {  
    for (int coluna = 0; coluna < num[linha].length; coluna++) {  
        num[linha][coluna] = teclado.nextInt();  
    }  
}
```

## ❑ Escrita:

```
for (int linha = 0; linha < num.length; linha++) {  
    for (int coluna = 0; coluna < num[linha].length; coluna++) {  
        System.out.println(num[linha][coluna]);  
    }  
}
```

# CAP. V. Arrays - Multidimensional

- ❑ EX: Programa em Java que armazena numa matriz 2x2 os 4 primeiros numeros inteiro digitado, e de seguida exhibe os números na tela.

## Resolução

```
public class Fundamentos {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int num[][] = new int[2][2];

        /*Ler os números */
        for (int linha = 0; linha < num.length; linha++) { //ciclo da linha
            for (int coluna = 0; coluna < num[linha].length; coluna++){ //ciclo da coluna
                System.out.println("Digite o número: ");
                num[linha][coluna] = teclado.nextInt();
            }
        }

        /*mostrar os números armazenados na matriz */
        for (int linha = 0; linha < num.length; linha++) { //ciclo da linha
            for (int coluna = 0; coluna < num[linha].length; coluna++){ //ciclo da coluna
                System.out.println(num[linha][coluna]);
            }
        }
    }
}
```



# CAP. V. ArrayList

- ❑ Os arrays(Vectores e Matrizes) possuem um tamanho fixo, ou seja não alteram o seu em tempo de execução.
- ❑ ArrayList: é uma class de colecção em java que permite armazenar diversos valores de forma dinâmica.
- ❑ Sintaxe:

```
ArrayList <Tipo> nome_do_ArrayList = new ArrayList< Tipo>();
```

- **Tipo:** é o tipo do ArrayList, não pode ser tipo primitivo, mas pode ser um tipo por referencia(String, Integer, Float, Double, Character, entre outros)
- **nome\_do\_ArrayList:** corresponde ao nome da variavel que armazenará o conjunto de valores do tipo definido.

## ❑ Exemplo

```
ArrayList <String> nomes = new ArrayList<String>(); // ArrayList de String  
ArrayList <Double> pesos = new ArrayList<Double>(); // ArrayList de float  
ArrayList <Integer> idades = new ArrayList<Integer>(); // ArrayList de int
```

# CAP. V. ArrayList – Métodos

- ❑ Para a inserir bem como para remover elementos existente em um ArrayList faz-se recursos de métodos:

```
ArrayList <String> nomes = new ArrayList();
```

Métodos	Descrição	Exemplo
add()	Adiciona um novo elemento no fim ou numa posição do ArrayList.	<code>nomes.add("Nsimba");</code> <code>nomes.add("Gabriel");</code> <code>nomes.add(0,"Malengue");</code>
set()	Altera o elemento encontrado na posição especificada por um novo elemento.	<code>nomes.set(1,"Mariano");</code>
contains()	Retorna true se ArrayList contiver o elemento, caso não retorna false.	<code>nomes.contains("Nsimba");</code>
get()	Retorna o elemento localizado no índice especificado.	<code>nomes.get(1);</code>
indexOf()	Retorna o índice da primeira ocorrência do valor especificado.	<code>nomes.indexOf("Gabriel");</code>
remove()	Remove a primeira ocorrência do valor especificado ou o elemento localizado no índice especificado.	<code>nomes.remove(1);</code> <code>nomes.remove("Malengue");</code>
size()	Retorna o numero de elementos armazenados.	<code>nomes.size();</code>

# CAP. V. ArrayList – Manipulação

## ❑ Adicionar elementos- add():

```
ArrayList <String> nomes = new ArrayList();  
nomes.add("Nsimba");  
nomes.add("Anacleto");  
nomes.add(0, "Malengue");
```

## ❑ Alterar elementos- set():

```
nomes.set(0, "Lukau");  
nomes.set(1, "Paulino");  
nomes.add(2, "Kondo");
```

## ❑ Imprimir elementos – get():

```
for (int i = 0; i < nomes.size(); i++) {  
    System.out.println(nomes.get(i));  
}
```

```
for (String nome : nomes ) {  
    System.out.println( nome);  
}
```

OU:

- ❑ A utilização da colecção ArrayList requer uma importação previa:
  - `import java.util.ArrayList;`

# CAP. V. ArrayList – Manipulação

- ❑ EX: Programa em Java que lê infinitamente números digitados pelo utilizador e mostra na tela. O programa termina se for digitado o numero -1.

## Resolução

```
import java.util.Scanner;
import java.util.ArrayList;
public class Fundamentos {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        ArrayList<Integer> numeros = new ArrayList<>();
        int num;
        while (true) {
            System.out.println("Digite um numero: ");
            num = input.nextInt();
            if (num == -1)
                break;
            else
                numeros.add(num);
        }

        for (Integer numero : numeros) {
            System.out.println( numero );
        }
    }
}
```

# CAP. V. Arrays - Exercícios

1. Faça um programa que recebe uma quantidade de números determinados pelo utilizador e imprime-os de forma inversa.
2. Faça um programa em Java que lê 20 números inteiros e mostra o maior e o menor.
3. Faça um programa que lê 30 números e mostra o vector ordenado em forma decrescente.

# Conteúdo



Cap. VI

String

# CAP. VI - String

- ❑ Em Java existe o recurso do tipo **String** para representar o conjunto de caracteres. Não como um tipo de dado primitivo mas como uma referência (objecto).
- ❑ **String** é um tipo de dado que permite armazenar um conjunto de caracteres.
- ❑ **Características**
  - Imutáveis: não se podem modificar
  - São referências
  - Contêm operações

# CAP. VI. String - Declaração

- ❑ Por ser uma classe, para sua utilização é necessária a criação de um de um objecto do tipo **String**.

- ❑ Declaração 1: `String nomeDaString;`

- ❑ Declaração 2: `String nomeDaString = new String();`

Método construtor

## ❑ Inicialização:

```
String nomeDaString = new String(); //Inicializa com espaço em branco ("")  
String nomeDaString = null; //Inicializa com o valor default para os objectos  
String nomeDaString = new String("Ola"); //Inicializa com a String (Olá)  
String nomeDaString = "Ola mundo"; //Inicializa com a String (Olá mundo)
```



# CAP. VI - String

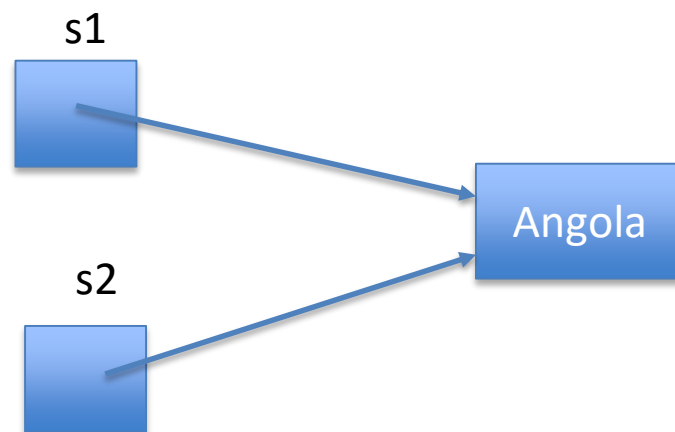
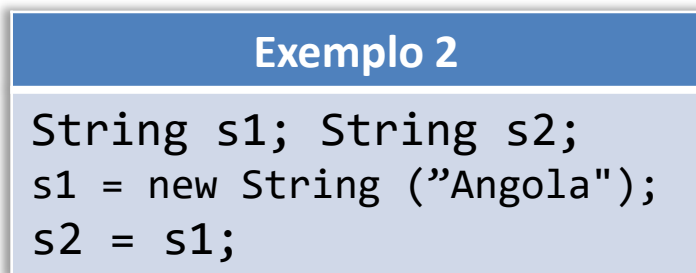
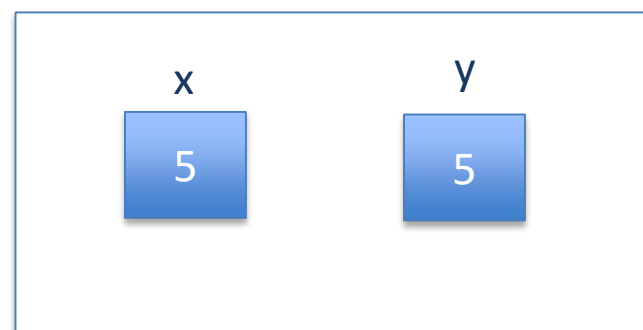
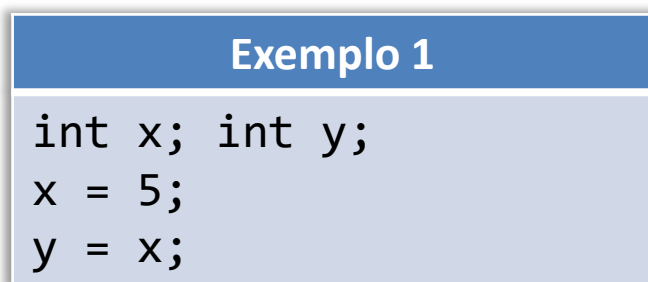
- ❑ A representação de uma String obedece a regra dos vectores de char. Ou seja cada caractere está localizado em determinada posição contando de 0 à n-1.
- ❑ EX: String s = "JAVA";

S =

J	A	V	A
0	1	2	3

# CAP. VI - String




- ❑ Uma das principais características que a **String** possui é a imutabilidade. Isto é, o conteúdo de uma **String** não altera por qualquer que seja a operação em que esteja envolvida.



- **s1** e **s2** possuem a mesma referência para o objecto.

# CAP. VI. String - Manipulação

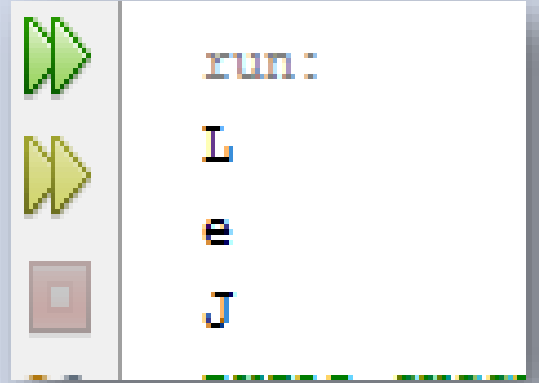
- ❑ Como toda a classe, a String possui um conjunto de métodos que facilitam a sua manipulação:
- ❑ **length()** – método que retorna o comprimento em número de caracteres de uma String.

Exemplo	Resultado
<pre>String s = "Linguagem Java"; System.out.println(<b>s.length()</b>);</pre>	   <pre>run: 14 BUILD SUCCESSFUL  </pre>

- **Nota:** `length()`  $\neq$  `length` (método associado à determinação da dimensão de uma matriz)

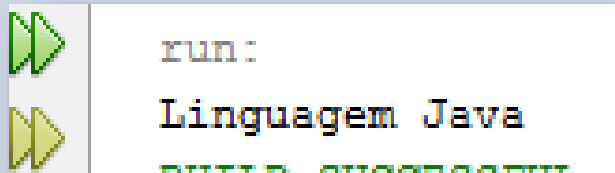
## CAP. VI. String - Obter caracteres individuais

- ❑ `charAt()` - método que retorna um caractere em uma determinada posição.

Exemplo	Resultado
<pre>String s = "Linguagem Java";  System.out.println(s.charAt(0)); System.out.println(s.charAt(7)); System.out.println(s.charAt(10));</pre>	

# CAP. VI. String - Concatenação






❑ **concat()** - método que retorna a concatenação (união) de duas Strings.

Exemplo	Resultado
<pre>String s1 = "Linguagem "; String s2 = "Java"; String s3 = s1.concat(s2); System.out.println(s3);</pre>	 <pre>run: Linguagem Java</pre>

**Nota:** Pode-se também concatenar usando o operador (+): **String s3 = s1 + s2;**

# CAP. VI. String – Localização Ocorrência

- ❑ **indexOf()** – método que retorna a posição da primeira ocorrência de um caracter ou uma sequencia de caracteres dentro de uma String. Retorna -1 se não localizar a ocorrência.

Exemplo: Inicio -Fim	Resultado
<pre>String s = "Linguagem Java"; System.out.println(s.indexOf('n')); System.out.println(s.indexOf('a', 7)); System.out.println(s.indexOf("gem")); System.out.println(s.indexOf("JAVA"));</pre>	 run:  2  11  6  -1

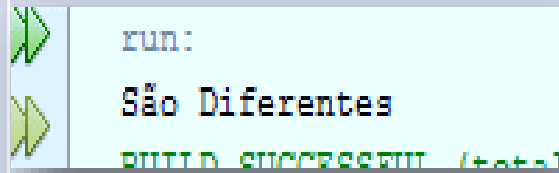
- **s.indexOf('n')** – retorna **2** ( localização do caracter **n** na string **s** ).
- **s.indexOf('a', 7)** – retorna **11** ( localização do caracter **a** na string **s** começando pela posição 7 ).
- **s.indexOf("gem")** – retorna **6** ( localização inicial da sequência **gem** na string **s** ).
- **s.indexOf("JAVA")** – retorna -1 (sequência **JAVA** não localizada na string **s** ).

- ❑ **O método lastIndexOf()** – semelhante ao anterior mas este obtém a posição da ultima ocorrência, fazendo a pesquisa do fim ao começo da string.

# CAP. VI. String - Manipulação

❑ **equals()** e **equalsIgnoreCase()** – compara duas Strings e retorna **true** se as duas Strings forem iguais ou **false** caso as Strings sejam diferentes.


- **equals()** - considera maiúsculas e minúsculas na comparação.
- **equalsIgnoreCase()** - ignora o facto de maiúsculas ou minúsculas.

Exemplo	Resultado
<pre>String s1 = "Linguagem Java"; String s2 = "Linguagem JAVA";  if (s1.equals(s2))     System.out.println("São Iguais"); else     System.out.println("São Diferentes");</pre>	 <pre>run: São Diferentes BUILD SUCCESSFUL (total</pre>

- Nota: **s1 == s2** – faz comparação entre referências, Não deve ser usado para saber se duas string são iguais.

# CAP. VI. String - Comparação

❑ **compareTo()** e **compareToIgnoreCase()** – são dois métodos que permitem comparar duas Strings.

- **compareTo** - considera maiúsculas e minúsculas na comparação.
- **compareToIgnoreCase** - ignora o facto de maiúsculas ou minúsculas.
- **s1.compareTo(s2)** 
  - s1 < s2 - retorna um valor negativo
  - s1 = s2 - retorna zero
  - s1 > s2 - retorna um valor positivo

Exemplo	Resultado
<pre>String s1 = "Java"; String s2 = "Fundamentos"; System.out.println("s1.compareTo(s2): " + s1.compareTo(s2)); System.out.println("s2.compareTo(s1): " + s2.compareTo(s1));</pre>	<pre>s1.compareTo(s2): 4 s2.compareTo(s1): -4</pre>



# CAP. VI. String - Conversão

- ❑ **toUpperCase()**- retorna uma nova String com caracteres em maiúsculo.
- ❑ **toLowerCase()**- retorna uma nova String com caracteres em minúsculo.

Exemplo	Resultado
<pre>String s1 = "agora"; String s2 = "JAVA"; System.out.println(s1 + " - Covertiu-se em: " + s1.toUpperCase()); System.out.println(s2 + " - Covertiu-se em: " + s2.toLowerCase());</pre>	<pre>agora - Covertiu-se em: AGORA JAVA - Covertiu-se em: java</pre>

# CAP. VI. String - Sub-cadeias

- ❑ **substring()** - retorna parte de uma String dependendo do intervalo.

Exemplo	Resultado
<pre>String s = "Linguagem Java"; System.out.println("1: " + s.substring(0,8)); System.out.println("2: " + s.substring(5));</pre>	<pre>1: Language 2: agem Java</pre>

- **s.substring(0,8)** – obtém do caracter 0 até ao caracter 8 não inclusive.
- **s.substring(5)** – obtém do caracter 5 até ao fim.

# CAP. VI. String - Exercícios

1. Implemente um programa em Java para ler uma String e diga se a mesma é capicua. Uma String é capicua se a String original for igual a sua inversa. Ex: ana – inverso: ana; ovo – inverso ovo.
2. Implemente um programa que lê duas Strings e imprime a quantidade de vezes que cada caractere da primeira ocorre na segunda.
3. Implemente um programa que lê uma String, e imprime a String resultante das seguintes operações e a quantidade de substituições caso ocorra:
  - 'a' substitui por 't'
  - 'e' substitui por 'h'
  - 'i' substitui por 'a'