<LAB365>

JAVA BÁSICO - PARTE 1

<LAB365> SENAI

AGENDA

- Revisão
- Introdução
- Vetores (Arrays)
- Matrizes
- ArrayList

REVISÃO

INTRODUÇÃO

- **Objetivos**
  - Apresentar os conceitos de vetores (arrays), matrizes e ArrayList em Java.
  - Demonstrar como declarar, inicializar e acessar elementos de vetores e matrizes.
  - Ensinar as principais operações com vetores e matrizes, como percorrer, buscar e modificar elementos.
  - Introduzir o conceito de ArrayList e suas vantagens em relação aos vetores.
  - Apresentar exemplos práticos de utilização de vetores, matrizes e ArrayList em Java.

VETORES

- **Variável**
  - Analogia: uma caixa, na qual você pode dar o nome que lhe achar conveniente, e guardar o conteúdo que desejar
  - Possui um tipo (String, boolean, int, double, ou qualquer classe...)
  - O valor dentro da “caixa” que pode ser alterado de acordo com a execução do programa

VETORES

- Agora imagine como ficaria na declaração de variáveis, declarando uma a uma, as 50 variáveis para o nome, depois as variáveis para as médias de cada aluno...

VETORES

- Em casos como esse que é útil a utilização da estrutura de dados conhecida como **vetor**
- Um vetor é uma espécie de caixa com várias divisórias para armazenar coisas (dados)
  - É uma variável que pode armazenar vários valores

nomes	André	Paulo	Maria	Davi	

VETORES

- A forma mais eficiente de trabalhar com coleções de elementos em Java é através da construção de vetores (arrays).
- Em Java, ***arrays são objetos*** ***que armazenam múltiplas variáveis do mesmo tipo***
  - ***Uma dimensão = vetor***
  - ***Duas dimensões = matriz***
- Uma vez criado, um ***array*** não pode ter seu tamanho alterado.

VETORES

- Os vetores são definidos pelo **tipo de dados** que eles devem armazenar e a **quantidade de posições**
- **Exemplo**:
  - Vetor de 8 posições para armazenar números reais
  - Vetor de 40 posições para armazenar objetos do tipo Aluno
- Os vetores são estruturas **homogêneas**.
  - Ex: um vetor de inteiros só armazena dados do tipo inteiro

VETORES

- **Criado Vetores - Arrays**
  - Sintaxe:
    - <tipo> [] <nome> = new <tipo>[tamanho];
  - Exemplos
    - int[] numeros = new int[3];
    - String[] nomes = new String[3];

VETORES

- **Criado e Inicializando Vetores - Arrays**
  - Sintaxe:
    - <tipo> [] <nome> = {valor0, valor1, valor2, ..., valorN};
  - Exemplos:
    - double[] notas = {4.5, 7.5, 8.0}

VETORES

- **Acessando os elementos do vetor**
  - Os elementos são acessados através do operador de indexação [].
  - O índice dos elementos inicia em zero.
  - A exceção I*ndexOutOfRangeException* é levantada se um índice inválido é usado.
- Exemplo:
  - System.out.println(“1º Nota: ” + notas[0]);

VETORES

- **Atribuindo valor a um elemento**
  - notas[0] = 7.5;
  - notas[1] = 5.5;
  - notas[2] = 10.5;
- A atribuição também pode ser realizada por inserção do usuário com um objeto Scanner.

VETORES

- **Tamanho de Arrays**
  - Para conhecer o tamanho total de um array basta você acessar o atributo *length*
  - Este atributo retorna um valor inteiro (*int*) que indica qual a capacidade máxima de armazenamento deste array

VETORES

- **Iterando por um vetor**
  - Iterar um array é percorrer todos os seus elementos.
  - Utiliza-se alguma estrutura de repetição para realizar a iteração.
  - for

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

VETORES

- **Iterando por um vetor**
  - foreach

```
for(double nota :  notas){
        System.out.println(nota);
}
```

VETORES

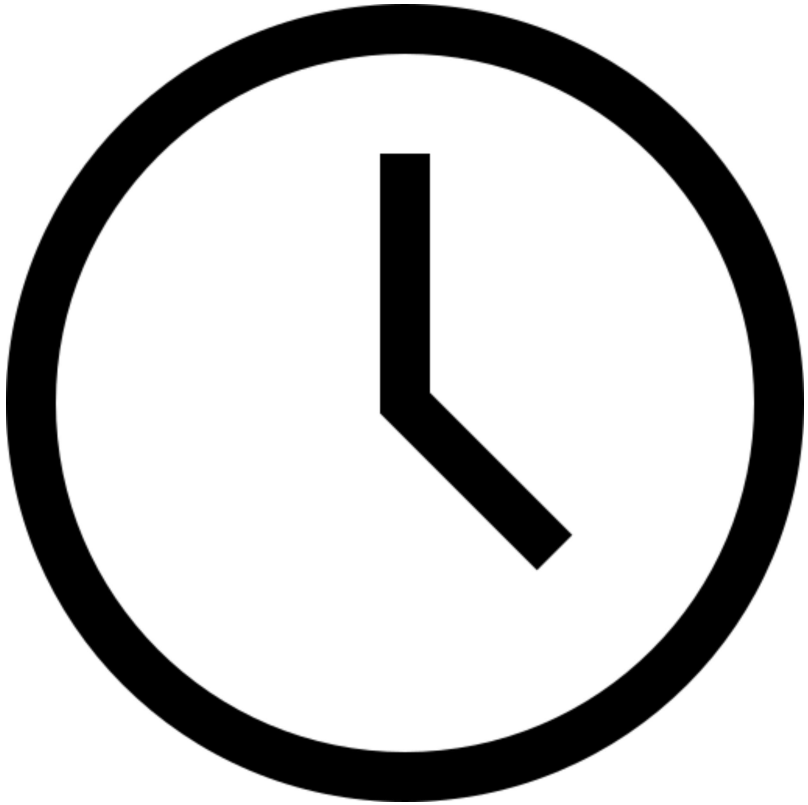
- Vamos desenvolver um pequeno algoritmo que cadastre uma quantidade determinada de alunos e suas respectivas médias. Em seguida iremos determinar qual a maior e menor média e a média da turma.

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



MATRIZES

- Uma matriz é uma estrutura comumente denominada de vetor de vetores. Tendo duas dimensões, linha e coluna, mantendo uma estrutura homogênea.
- **Declaração de uma matriz**
  - <tipo> [][] <nome> = new <tipo>[tamanho_linha][tamanho_coluna];
  - int[][] matriz = new int[2][3];
  - matriz.length; //retorna a quantidade de linhas
  - matriz[0].length; //retorna a quantidade de colunas

MATRIZES

- Iterando uma matriz
  - Para iterar uma matriz precisamos discorrer das suas linhas e colunas. Com isso iremos utilizar dois laços de repetição.

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

MATRIZES

- Iterando uma matriz
  - foreach

```
for (int[] vetor : matriz) {
        for (int elemento : vetor) {
        System.out.println(elemento);
        }
}
```

MATRIZES

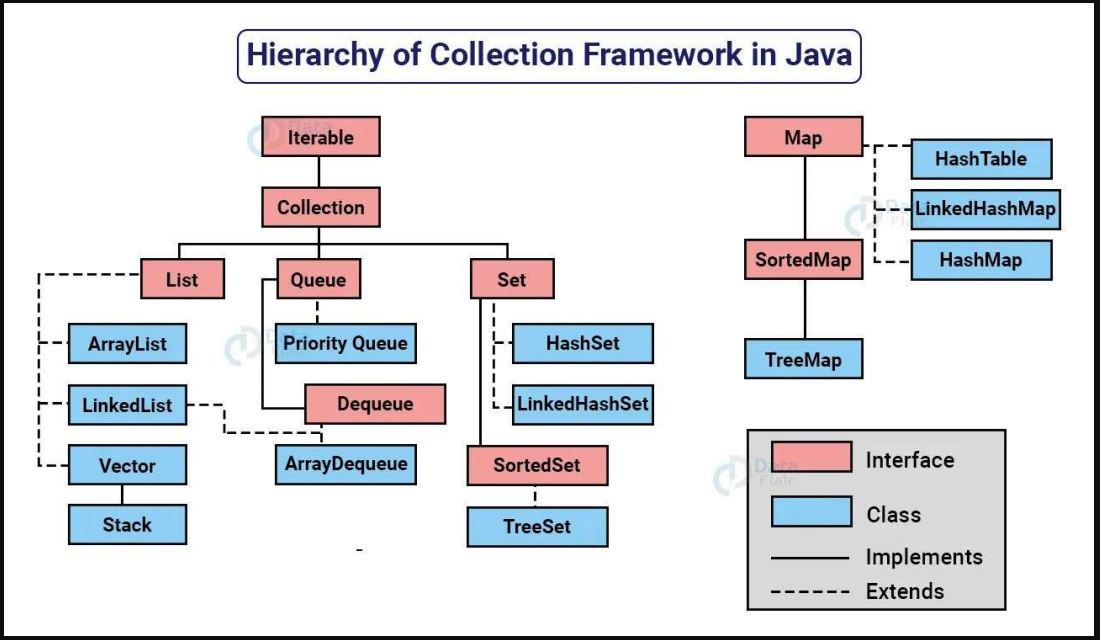
- Faça um algoritmo que preencha uma matriz 5x5 de inteiros e escreva:
  - a) a soma dos números ímpares fornecidos;
  - b) a soma de cada uma das 5 colunas;
    c) a soma de cada uma das 5 linhas.

ARRAYLIST

- A Java API fornece várias estruturas de dados pré-definidas, chamadas coleções, usadas para armazenar grupos de objetos relacionados na memória. Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem a necessidade de conhecer como os dados são armazenados. Isso reduz o tempo de desenvolvimento de aplicativos.

ARRAYLIST

- A utilização das Coleções de objetos (Collections) disponibiliza diversas classes que representam algumas estruturas de dados.
- As principais estruturas são:
  - Listas (List)
  - Conjuntos (Sets)
  - Mapas (Maps)



ARRAYLIST

- O ArrayList é bastante semelhante ao vetor, porém, **não é um vetor**.
- Recebe um conjunto de **Objects**.
- Você já usou arrays para armazenar sequências de elementos. Arrays não mudam automaticamente o tamanho em tempo de execução para acomodar elementos adicionais.
- Telefone[] telefones = new Telefone[2];

ARRAYLIST

- A classe de coleção **ArrayList<T>** (pacote java.util) fornece uma solução conveniente para esse problema — ela pode alterar *dinamicamente* seu tamanho para acomodar mais elementos. O T (por convenção) *é um espaço reservado* — ao declarar um novo ArrayList, substitua-o pelo tipo dos elementos que você deseja que o ArrayList armazene.
- ArrayList<Integer> integers;

ARRAYLIST

- Para criar um ArrayList:
  - ArrayList<String> nomes= new ArrayList();
- A lista se adequa à medida em que vamos adicionando novos elementos.
- Para adicionar um novo elemento na lista utilizamos o método *add*(Object e);

ARRAYLIST

- ArrayList<Integer> dias = new ArrayList();

  dias.add(1);

  dias.add(2);

  dias.add(3);

  dias.add(4);

ARRAYLIST

- Podemos manipular os elementos da lista utilizando os seguintes métodos:
  - Get: para recuperar um Object de um determinado índice.
  - Remove: retira um elemento da lista e reorganiza ela.
  - Contains: verifica se um Object passando como parâmetro existe na lista.
  - Size: informa o tamanho da lista.
- *dias.get(1);*

  dias.remove(3);

  dias.contains(5);

🙌 Mão na massa...

ARRAYLIST

- Crie uma classe TesteArrayListNumero que possui um método main.
- Dentro do main crie um ArrayList de Integer.
- Adicione 10 números informados pelo usuário.
- Se o usuário tiver digitado os números 10, 100 ou 1000 mostre uma mensagem informando que ele ganhou um bônus de R$ 50,00

ARRAYLIST

- Crie uma classe TesteArrayListString que possui um método main.
- Dentro do main crie um ArrayList de String (lista1).
- Adicione 10 Strings informadas pelo usuário.
- Percorra a lista verificando se o usuário digitou alguma String com menos de 3 caracteres. Em caso positivo, adicione essa String em outra lista que você vai criar (lista2).
- Utilizando o método removeAll, remova todos os elementos dessa segunda lista (lista2) da lista principal (lista1).
- No final imprima a quantidade de Strings da lista.

<LAB365>

JAVA BÁSICO - PARTE 2

<LAB365> SENAI

AGENDA

- Funções

FUNÇÕES

- São conhecidas como:
  - Funções
  - Métodos
  - Procedimentos
  - Ações de execução

FUNÇÕES

O que são?

- São parte executáveis de código, essa parte é a menor unidade de execução de código. Ou seja, são como um “parágrafo de código”. Podemos ter procedimentos simples ou complexos.
- Cada método deve ter um nome descritivo.
- **Procedimentos simples**: exibir um valor, realizar um cálculo ou atribuir um valor.
- **Procedimentos complexos**: como ações compostas ou integrações.

FUNÇÕES

- **Sintaxe**

```
modificador retorno identificador(argumentos) {
        corpo
}
```

Modificador: Indica a visibilidade do método por outras classes.

Retorno: Tipo de dado que o método retornará.

Identificador: Nome do método.

Argumentos: São os parâmetros que o método precisa receber para ser executado.

Corpo: Onde fica a lógica implementada.

Boas Práticas

- Um método que realize uma tarefa muito complexa, deve ser dividido em outros métodos que realizam partes dessa tarefa. Assim, juntando tudo resolvem a tarefa principal.

- Com isso, a compreensão do método fica mais fácil, tanto para os demais desenvolvedores quanto para o mesmo que desenvolveu.

- Exemplo: Ao realizar uma longa viagem de ônibus do pontoA para o pontoB. A tarefa complexa é "realizar a viagem", porém há diversas coisas que precisam ser executadas para que o resultado ocorra.

FUNÇÕES

Boas práticas - Exemplo

```
viajarOnibus(pontoA, pontoB) {
    ir(pontoA, pontoB);
    aproveitar();
    voltar(pontoA, pontoB);
}

ir(pontoA, pontoB) {
    organizarHorarios();
    comprarPassagem(pontoA, pontoB);
    reservarHotel(pontoB);
    organizarItinerario()
    fazerMalas();
    embarcar();
    aguardarChegada();
    irHotel();
    desfazerMalas();
}
```

```
aproveitar() {
    consultarItinerarioDia();
}

voltar(pontoA, pontoB) {
    organizarHorarios();
    comprarPassagem(pontoB, pontoA);
    fazerMalas();
    embarcar();
    aguardarChegada();
    irCasa();
    desfazerMalas();
}
```

FUNÇÕES

Retorno

- Indica o tipo de dado que a função retorna.
- Pode ser um tipo primitivo, não primitivo ou sem retorno ("vazio"):
    - Para indicar que a função não tem retorno, utilize a palavra-chave "void";
    - Para que a função retorne a informação necessária, utilize a palavra-chave "return" seguida da variável ou valor a ser retornado.

FUNÇÕES

Retorno - Exemplos

```
// Sem retorno (retorno vazio)
public void setDataAtual() {
      this.dataAtual = new Date();
}

// Com retorno (retorno do tipo "java.util.Date")
public Date getDataAtual() {
      return this.dataAtual;
}
```

FUNÇÕES

Identificador

- Nome da função.

- Boas práticas:
  - Utilizar identificadores descritivos, que indiquem o que a função faz;
  - Para a nomenclatura, o padrão "camelCase" é usado;
  - Funções que inserem valores devem usar prefixo com "set" seguido do nome do atributo. Ex: "*setNome*";
  - Funções que recuperam valores devem usar prefixo com "get" seguido do nome do atributo Ex: "*getNome*";

Argumentos

- Conhecidos também como parâmetros das funções.
- São utilizados no corpo da função como variáveis locais.
- Os argumentos são opcionais.

FUNÇÕES

Argumentos - exemplos

```java
// Sem argumento
public void escreverConsoleDataAtual() {
	System.out.println(new Date());
}

// Com argumento do tipo "java.util.Date"
public void escreverConsoleData(Date data) {
	System.out.println(data);
}

// Com argumento e verificação se número é par
public boolean numeroPar(int numero) {
	int mod = numero % 2;
	boolean retorno = mod == 0;
	return retorno;
}
```

FUNÇÕES

Corpo

- O corpo da função é onde a lógica é implementada.
- Nele é possível criar novas variáveis, executar operações aritméticas, percorrer laços de repetição, dentre outros procedimentos necessários para que a função seja executada corretamente.

FUNÇÕES

🙌 Mão na massa...

FUNÇÕES

Criar uma função que calcule sua idade.

Agora, vamos melhorar nossa aplicação criando métodos para:

- Pegar ano atual do sistema;
- Pedir ao usuário o ano do seu nascimento.

FUNÇÕES

Os retornos das funções podem ser:

- tipos primitivos (int, double, boolean, char, ...);
- tipos não primitivos (objetos);
- vazio (void).

FUNÇÕES

Os retornos podem ser armazenados em variáveis ou até mesmo serem utilizados diretamente.

```java
public void exibirDataAtual() {
    Date dataAtual = getDataAtual();
    System.out.println("Hoje é: " + dataAtual);
}

private Date getDataAtual() {
    return new Date();
}
```

```java
public void exibirDataAtual() {
    System.out.println("Hoje é: " + getDataAtual());
}

private Date getDataAtual() {
    return new Date();
}
```

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

<LAB365>

<LAB365> SENAI