

Disciplina: PROGRAMAÇÃO I

Aula 10: Tratamento de dados homogêneos

I Apresentação

Trabalharemos com estruturas básicas de armazenamento de dados em vetores e matrizes, tanto de tipos de dados quanto de objetos. Abordaremos também a aplicabilidade de ArrayLists ao trabalharmos com dados armazenados em tabelas de banco de dados.

| Objetivo

- Identificar o uso de vetor de tipos primitivos e de vetor de objetos;
- Escrever programas com vetor de tipos primitivos;
- Desenvolver aplicações com ArrayList.

| Armazenamento de dados

É muito comum que nossos sistemas precisem armazenar uma quantidade grande de dados em memória. Isso ocorre em situações em que muitos dados precisam ser analisados em conjunto, e quando o armazenamento em variáveis comuns seria inviável. Essa inviabilidade se dá em função da enorme quantidade de variáveis, dificultando a entrada desses dados e as operações a serem realizadas com eles.

Analisar 1000 temperaturas e determinar a sua média é relativamente simples, uma vez que precisamos de apenas uma variável para a entrada de dados e uma variável totalizadora para realizar o somatório. Entretanto, se fosse necessário analisar quantas temperaturas são iguais ou maiores do que a média, seria muito mais complexo, pois precisaríamos armazenar na memória todos os valores das temperaturas, sendo necessárias 1000 variáveis.

Os vetores e matrizes nos ajudam muito nessas condições. Em Java, podem ser usados tanto para valores (tipos primitivos) como para objetos. Isso mesmo: podemos criar vetores e matrizes de objetos. Existem ainda várias outras opções de armazenamento de dados em memória em Java, mas **o ArrayList é um dos mais importantes, pois pode ser usado em conjunto com bases de dados.**



 Fonte: freepik

| Vetores

Primeiro vamos falar sobre os vetores. Eles são a forma mais simples de armazenarmos uma grande quantidade de dados de **mesmo tipo (homogêneo) e de tamanho fixo (deve-se determinar a quantidade em sua declaração)**. São ainda um conjunto de variáveis do mesmo tipo armazenados em sequência na memória. Todas essas variáveis possuem o mesmo nome, mas são diferenciadas através de um índice. Isso facilita muito a entrada de dados e o acesso aos elementos do vetor. Em uma variável convencional, usamos apenas o seu identificador.

Veja um exemplo a seguir:

```
import java.util.Scanner;

public class Vetores {
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        double temperatura;
        System.out.println("Digite a temperatura:");
        temperatura = Double.parseDouble(ent.nextLine());
        System.out.println("A temperatura é: " + temperatura);
    }
}
```

A entrada de uma variável para o armazenamento de uma variável é simples como visto no exemplo anterior, mas a criação de 1000 variáveis para armazenar as 1000 temperaturas seria inviável. Entretanto, podemos realizar facilmente essa tarefa com o auxílio de um vetor.

Exemplo com uso de um vetor:

```
import java.util.Scanner;

public class Vetores {
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        double temperatura[] = >new double[1000];
        for (int i = 0; i < 1000; i++) {
            System.out.println("Digite a temperatura: [" + (i + 1) + "]");
            temperatura[i] = Double.parseDouble(ent.nextLine());
        }
        System.out.println("Foram entradas as temperaturas:");
        for (int i = 0; i < 1000; i++) {
            System.out.println("Temperatura: [" + (i + 1) + "] = " + temperatura[i]);
        }
    }
}
```

Para realizar os testes, altere o valor de 1000 para 10, permitindo a análise dos resultados com uma quantidade menor de dados.

Perceba que podemos trabalhar com uma grande quantidade de dados em memória, mas com uma aplicação simples, bem como trabalhar com uma única variável.

Internamente, o nome do vetor identifica sua posição de memória, e o índice identifica a posição do elemento dentro do vetor. Por isso, devemos trabalhar cada elemento do vetor e não o vetor em si.

Exibir o vetor não é correto: o resultado será o endereço de memória do vetor e não o seu conteúdo:

```
double temperatura[] = new double[1000];
System.out.println("Temperaturas: " + temperatura);
```

O resultado será:

Temperaturas: [D@28d93b30

Será exibido o endereço de memória do vetor e não o conteúdo dos seus elementos.

Características

Os vetores possuem as seguintes características:


- 1

Devem ser homogêneos (todos os elementos são do mesmo tipo);
- 2

Possuem um tamanho fixo (uma vez criados, não podem aumentar o diminuir de tamanho);
- 3

O primeiro elemento de um vetor na linguagem Java é sempre 0 (zero);
- 4

O último elemento de um vetor em Java é dado pelo seu tamanho – 1;

[📄 Declaração de vetores de tipos básicos \(primitivos e String\) e de objetos](#) Clique no botão acima.

Declaração de vetores de tipos básicos (primitivos e String)

Não precisamos de muitos cuidados para a criação de vetores de tipos básicos, como podemos ver nos exemplos a seguir. Basta declarar o vetor e já podemos armazenar e recuperar os dados diretamente.

Exemplos:

```
byte binarios[] = new byte[20];
    short valores1[] = new short[20];
    int valores2[] = new int[20];
    long valores3[] = new long[20];
    float notas[] = new float[40];
    double temp[] = new double[1000];
    char letras[] = new char[26];
    boolean logicos[] = new boolean[20];
    String nomes[] = new String[10];
    binarios[0] = 0;
    binarios[19] = 0;
    valores1[0] = 0;
    valores1[19] = 0;
    valores2[0] = 0;
    valores2[19] = 0;
    valores3[0] = 0;
    valores3[19] = 0;
    notas[0] = 0;
    notas[39] = 0;
    temp[0] = 0;
    temp[999] = 0;
    letras[0] = 'A';
    letras[25] = 'Z';
    logicos[0] = false;
    logicos[19] = true;
    nomes[0] = "Maria";
    nomes[9] = "Marcela";
```

Foram criados vetores de todos os tipos básicos e armazenados valores diretamente no primeiro e último elemento de cada um dos vetores, sem a necessidade de algum tipo de cuidado maior quanto ao seu uso.

Observação: Tentar acessar um elemento inexistente de um vetor gerará uma exceção do tipo:

ArrayIndexOutOfBoundsException

Exemplo:

```
valor[0] = 1;
valor[99] = 100;
valor[100] = 101;
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 100 at Vetores.main(Vetores.java:10)
```

Sintaxe padrão para a declaração de vetores de tipos básicos:

```
<tipo> <nome>[] = new <tipo>[<num_elementos>];
```

O uso de vetores segue três passos:

- 1 Declaração do vetor: `int vetor[];`
- 2 Determinação do tamanho do vetor: `vetor = new int[40];`
- 3 Uso do vetor: `vetor[4] = 45;`

Pode-se também definir um vetor através da definição inicial dos valores dos seus elementos:

Exemplo:

```
float nota[] = {10.0f, 5.0f, 6.6f, 8.0f, 7.0f, 7.8f, 8.4f, 4.2f, 1.8f, 6.4f};
```

Será criado um vetor de reais (float) com 10 elementos e valores:

```
nota[0] = 10.0; nota[1] = 5.0; nota[2] = 6.6; nota[3] = 8.0; nota[4] = 7.0;
```

```
nota[5] = 7.8; nota[6] = 8.4; nota[7] = 4.2; nota[8] = 1.8; nota[9] = 6.4.
```

Declaração de vetores de objetos

Para vetores de objetos, devemos tomar alguns cuidados. Vetores de objetos na verdade são vetores que armazenam endereços de memória dos objetos (referências implícitas). Um objeto, ao ser declarado, não o cria, sendo necessário instanciá-lo para que possa ser usado. Uma declaração de um vetor de objetos apenas define o vetor e o seu tipo, mas não instancia (cria) os objetos. Dessa forma, cada objeto tido como elemento do vetor ainda não criado possui como endereço: null. Se tentarmos utilizar um objeto, que é um elemento de um vetor, sem instanciá-lo, teremos um lançamento de exceção: **NullPointerException**

Por isso, a declaração de um objeto...

```
Integer vetInteger[] = new Integer[15];
```

... define um vetor de objetos do tipo Integer, mas não instancia os objetos. Para isso, devemos criar cada um dos objetos que fazem parte do vetor:

```
for (int i=0; i<15; i++) {
    vetInteger[i] = new Integer(i);
}
```

Cada objeto do tipo Integer foi instanciado (criado) usando o valor de i como parâmetro no construtor, fazendo com que cada um tenha um valor associado durante a criação.

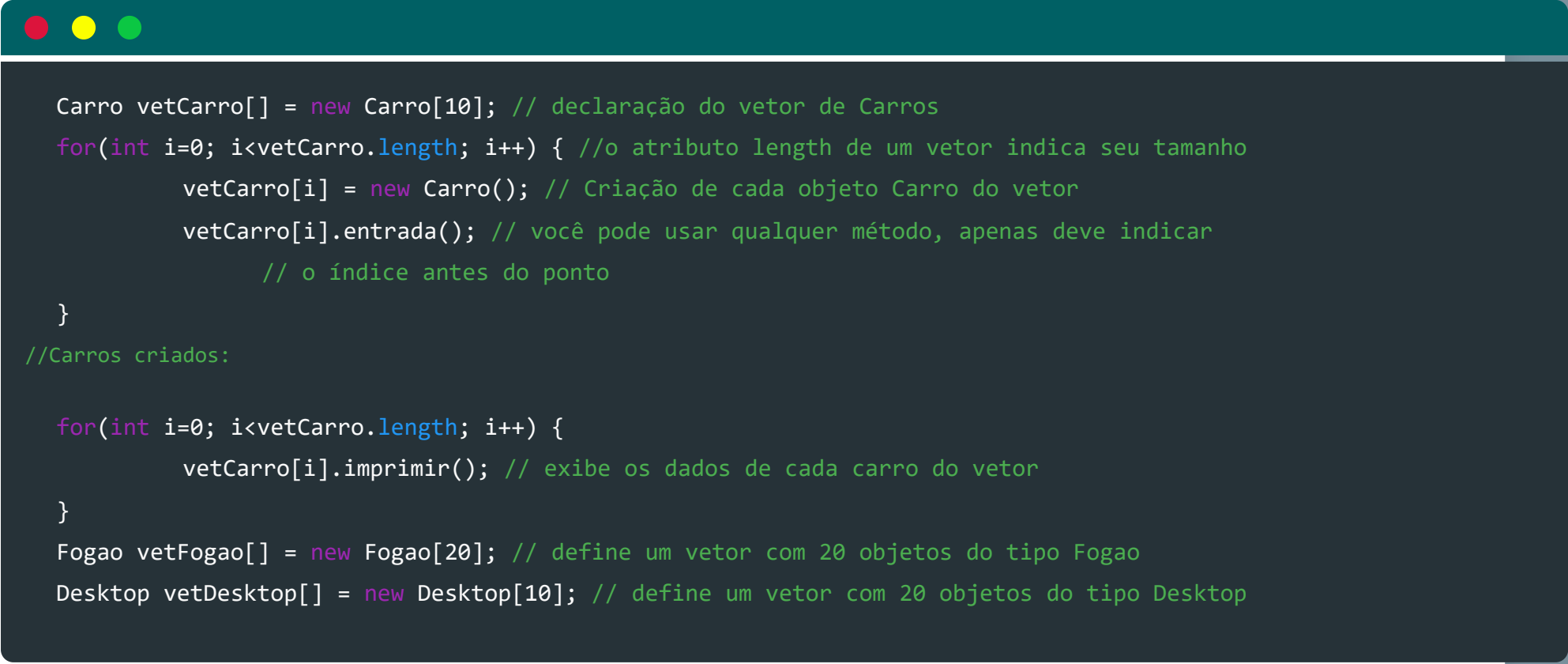
Para exibir os valores dos elementos do vetor devemos usar:

```
for (int i=0; i<15; i++) {  
    System.out.println("Posicao "+i+": " + vetInteger[i].intValue() );  
}
```

A conversão para intValue() se faz necessária porque um Integer é um objeto e não um tipo primitivo int. O mesmo serve para a classe Double, que será um objeto e não um tipo primitivo double.

Double e double são diferentes, o primeiro declara um objeto e o segundo um atributo ou variável do tipo primitivo double.

Podemos criar vetores de qualquer tipo, inclusive de objetos. Só não podemos esquecer de instanciar cada um dos elementos do vetor:



```
Carro vetCarro[] = new Carro[10]; // declaração do vetor de Carros  
for(int i=0; i<vetCarro.length; i++) { //o atributo length de um vetor indica seu tamanho  
    vetCarro[i] = new Carro(); // Criação de cada objeto Carro do vetor  
    vetCarro[i].entrada(); // você pode usar qualquer método, apenas deve indicar  
        // o índice antes do ponto  
}  
//Carros criados:  
  
for(int i=0; i<vetCarro.length; i++) {  
    vetCarro[i].imprimir(); // exibe os dados de cada carro do vetor  
}  
Fogao vetFogao[] = new Fogao[20]; // define um vetor com 20 objetos do tipo Fogao  
Desktop vetDesktop[] = new Desktop[10]; // define um vetor com 20 objetos do tipo Desktop
```




Fonte: unsplash

I Matrizes

Matrizes em Java seguem as mesmas regras que os vetores, sendo a grande diferença o fato de que uma matriz é bidimensional. Ou seja, não possui apenas uma dimensão, mas, duas dimensões e, por isso, dois índices. O primeiro, para definir a linha e, o segundo, para definir a coluna.

Veja a seguir um exemplo de matriz:

Matriz 5 x 5 de inteiros.

0	1	2	3	4
1	11	12	13	14
2	21	22	23	24
3	31	32	33	34
4	41	42	43	44

Declaração da matriz em Java:

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

A matriz também pode ser definida com seus valores na declaração. A matriz usada no exemplo poderia ser declarada da seguinte forma:

```
int matriz[ ][ ] = { {0, 1, 2, 3, 4}, {1, 11, 12, 13, 14}, {2, 21, 22, 23, 24},
```


{3, 31, 32, 33, 34}, {4, 41, 42, 43, 44} };

Cada linha deve ser definida dentro de um conjunto de chaves próprio.

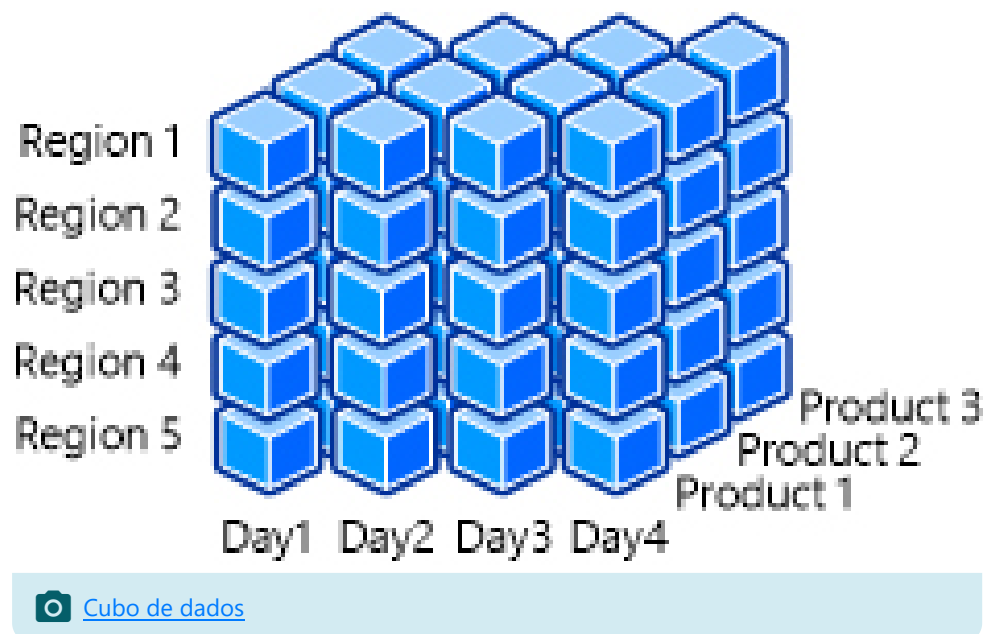
Matrizes, assim como os vetores, são homogêneas e possuem tamanho fixo.

A linguagem Java ainda permite a criação de estruturas com mais dimensões, como podemos ver nos exemplos a seguir:

```
double dados[ ][ ][ ] = new double[4][5][3];
```

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

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



Podemos criar matrizes de todos os tipos básicos, assim como de objetos.

Veja exemplos de matrizes de tipos básicos:

```
byte binarios[ ][ ] = new byte[10][20];
short valores1[ ][ ] = new short[20][20];
int valores2[ ][ ] = new int[30][20];
long valores3[ ][ ] = new long[30][20];
float notas[ ][ ] = new float[20][40];
double temp[ ][ ] = new double[50][1000];
char letras[ ][ ] = new char[10][26];
boolean logicos[ ][ ] = new boolean[15][20];
String nomes[ ][ ] = new String[20][10];
```

Podemos então atribuir diretamente os valores a cada elemento de uma matriz de tipos básicos:

```
binarios[0][0] = 0;
binarios[9][19] = 0;
valores1[0][0] = 0;
valores1[19][19] = 0;
valores2[0][0] = 0;
valores2[29][19] = 0;
valores3[0][0] = 0;
valores3[29][19] = 0;
```

```
notas[0][0] = 0;
notas[19][39] = 0;
temp[0][0] = 0;
temp[49][999] = 0;
letras[0][0] = 'A';
letras[9][25] = 'Z';
logicos[0][0] = false;
logicos[14][19] = true;
nomes[0][0] = "Maria";
nomes[19][9] = "Marcela";
```

Veja exemplos de matrizes de objetos:

```
public class AppVetor {
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        // declaração da matriz de Carros com 300 elementos
        Carro matCarro[][] = new Carro[10][30];
        // o atributo length de uma matriz, sem indicar a linha, indica o número de
        // linhas
        for (int i = 0; i < matCarro.length i++) {
            // o atributo length de uma matriz, com a indicação de uma linha,
            // indica o número de colunas
            for (int j = 0; j < matCarro[0].length; j++) {
                matCarro[i][j] = new Carro(); // Criação de cada objeto Carro do vetor
                matCarro[i][j].entrada(); // você pode usar qualquer método, apenas deve
                // indicar os índices antes do ponto
            }
        }
        // Carros criados:
        for (int i = 0; i < matCarro.length; i++) {
            for (int j = 0; j < matCarro[0].length; j++) {
                matCarro[i][j].imprimir(); // exibe os dados de cada carro da matriz
            }
        }

        // define uma matriz com 400 objetos do tipo Fogao
        Fogao matFogao[][] = new Fogao[20][10];matFogao[0][0]= new Fogao();
        matFogao[10][9]= new Fogao();
        // define uma matriz com 100 objetos do tipo Desktop
        Desktop matDesktop[][] = new Desktop[10][10];
        matDesktop[0][0] = new Desktop();
        matDesktop[9][9] = new Desktop();
    }
}
```


Como essas matrizes são de objetos, não podemos esquecer que é necessário instanciar cada elemento do vetor para podermos utilizar o objeto. Caso contrário, teremos uma exceção de: **NullPointerException**

```
Carro matCarro[][] = new Carro[10][30];
matCarro[i][j] = new Carro(); // Instanciação de cada objeto Carro da matriz
```

ArrayList

Um ArrayList é uma estrutura de dados baseada em um array (vetor), mas um ArrayList possui tamanho variável, em que é possível inserir elementos em qualquer parte do array. Um ArrayList é heterogêneo, ou seja, pode armazenar qualquer tipo de dados, como tipos primitivos, além de qualquer tipo de objetos.

Pode-se criar um ArrayList através de três diferentes construtores, são eles:

 Clique nos botões para ver as informações.

Sem parâmetros (vazio).



É criado um ArrayList com 10 elementos iniciais e, ao se inserir um novo elemento quando se atinge o limite, serão criados 10 novos espaços de endereçamento.
Exemplo:

```
ArrayList array = new ArrayList();
```

Com um parâmetro inteiro n



É criado um ArrayList com n elementos iniciais e, ao ser inserir um novo elemento quando se atinge o limite, serão criados 10 novos espaços de endereçamento.
Exemplo:

```
// Mil elementos iniciais e dez novas posições a cada necessidade  
ArrayList array = new ArrayList(1000);
```

Com dois parâmetros inteiros n e m



É criado um ArrayList com n elementos iniciais e, ao se inserir um novo elemento quando se atinge o limite, serão criados m novos espaços de endereçamento.
Exemplo:

```
// Mil elementos iniciais e cem novas posições a cada necessidade  
ArrayList array = new ArrayList(1000, 100);
```

Vamos analisar o exemplo a seguir:



Classe: AppArrayList.

```
import java.util.ArrayList;  
public class AppArrayList {  
    public static void main(String[] args) {
```

```
ArrayList array = new ArrayList();
array.add(35); // insere um inteiro
array.add(new Carro()); // insere um objeto carro
array.add(0, 3.75); // insere um double no início (índice 0)
array.add(new Fogao()); // insere um Fogão
array.add(2, "Maria"); // insere um String na posição 2
array.add(new Desktop()); // insere um objeto Desktop
System.out.println("A lista possui " + array.size() + " elementos.");
for (int i = 0; i < array.size(); i++) {
    System.out.println("Elemento[" + i + "]: " + array.get(i));
}
}
```


Resultado da execução:

```
A lista possui 6 elementos.
Elemento[0]:3.75
Elemento[1]:35
Elemento[2]:Maria
Elemento[3]:Carro@15db9742
Elemento[4]:Fogao@6d06d69c
Elemento[5]:Desktop@7852e922
```

Notas

- 1 Um ArrayList pode armazenar diferentes tipos, sejam eles básicos ou objetos;
- 2 Podemos inserir elementos em qualquer posição que o ArrayList é reorganizado;

Principais métodos de ArrayList

 Clique no botão acima.

Principais métodos de ArrayList

- `boolean add(Object element)`: responsável por adicionar novos elementos no final da lista;
- `void add(int index, Object element)`: responsável por inserir um novo elemento em uma determinada posição da lista;
- `void clear()`: responsável por limpar a lista, removendo todos os seus elementos;
- `boolean contains(Object element)`: responsável por realizar uma busca na lista por um determinado elemento. Retorna verdadeiro (true), caso o elemento seja encontrado; ou falso (false), caso o elemento não tenha sido encontrado;
- `Object get(int index)`: responsável por buscar e retornar um determinado elemento da lista, na posição indicada por index;
- `int indexOf(Object element)`: responsável por verificar e retornar a posição da primeira ocorrência de um determinado elemento na lista, que tenha sido especificado para busca na lista: elemento;
- `boolean isEmpty()`: responsável por verificar e retornar verdadeiro (true) caso a lista esteja vazia (sem elementos), e falso (false) caso a lista contenha elementos;
- `int lastIndexOf(Object element)`: responsável por verificar e retornar a posição da última ocorrência de um determinado elemento na lista, que tenha sido especificado para busca na lista: elemento;
- `Object remove(int index)`: responsável por remover um elemento indicado pelo índice (index) da lista;
- `Object set(int index, Object element)`: responsável por substituir um elemento indicado pelo índice (index) da lista por outro elemento;
- `int size()`: responsável por retornar a quantidade de elementos existentes na lista.

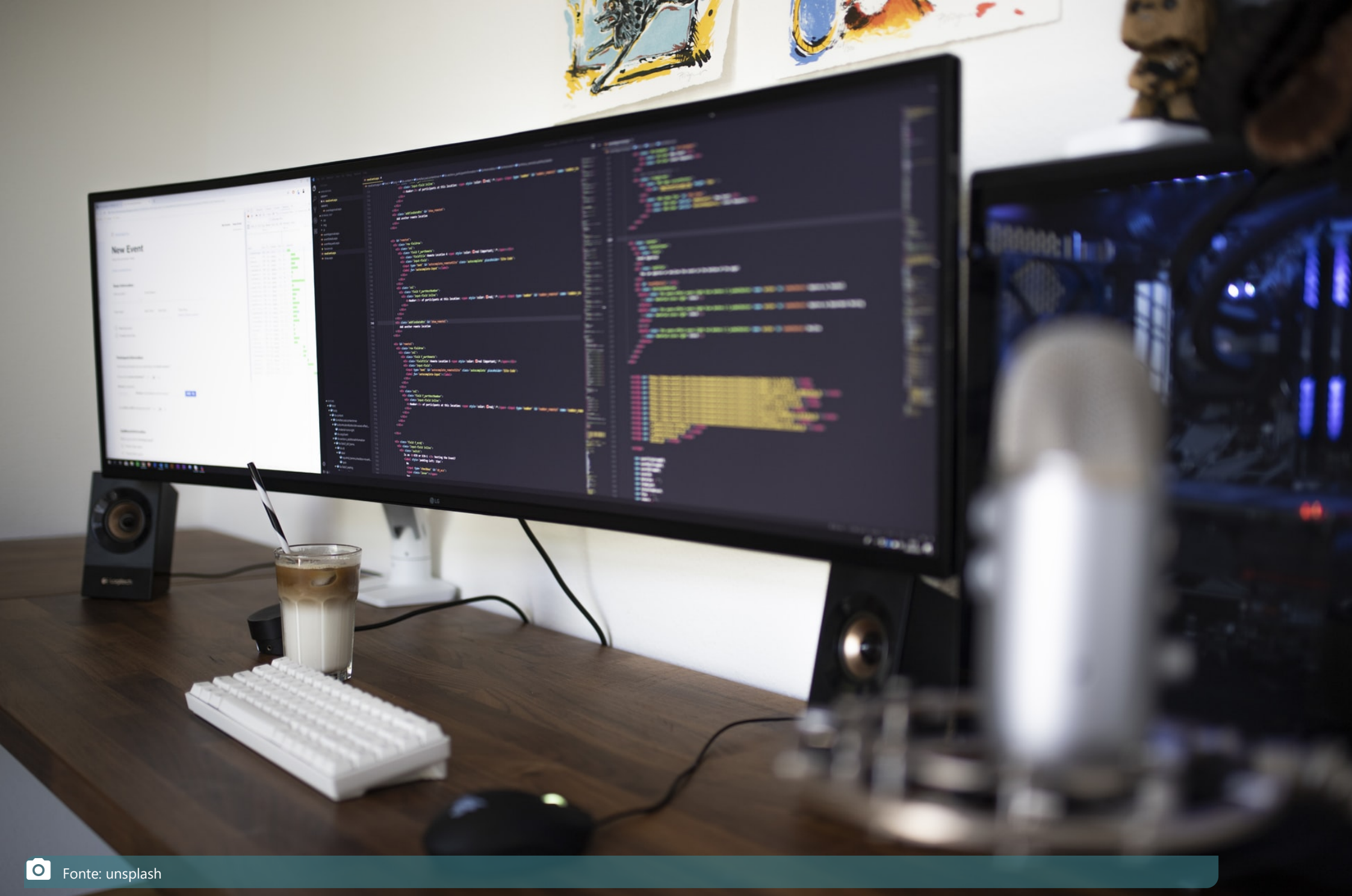
É muito comum criarmos ArrayLists de um tipo específico de objetos:

```
// ArrayList de objetos do tipo String
ArrayList<String> nomes = new ArrayList<String>();

// ArrayList de objetos do tipo Carro
ArrayList<Carro> carros = new ArrayList<Carro>();

// ArrayList de objetos do tipo Fogao
ArrayList<Fogao> fogoes = new ArrayList<Fogao>();

// ArrayList de objetos do tipo Desktop
ArrayList<Desktop> desks = new ArrayList<Desktop>();
```



Fonte: unsplash

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Iterator

Uma das principais utilidades de um ArrayList é o armazenamento de resultados de consultas à base de dados. O resultado da consulta é normalmente enviado para um ArrayList através de um ArrayAdapter. O ArrayList passa então a conter todos os registros retornados da consulta ao banco. Os dados de cada registro podem ser repassados a objetos, que são armazenados no ArrayList. Através do ArrayList, podemos realizar uma interação (iterator) para buscar os dados de cada elemento armazenado em um objeto e montar relatórios. A geração de relatórios é fundamental para todo e qualquer sistema, sendo primordial para análise dos usuários.

Um iterator nada mais é do que um objeto capaz de acessar sequencialmente um ArrayList, permitindo que cada elemento possa ser repassado a um objeto, que, por ventura, poderá disponibilizar seus dados para a montagem do relatório.

Sintaxe para a criação de um iterator:

```
Iterator<String> iterator = grupoTrabalho.iterator();
```

Acesso sequencial ao iterator:

// enquanto houver elementos no iterator, caso contrário sairá do laço

```
while (iterator.hasNext()) {  
    // busca o próximo elemento no iterator  
    System.out.println("Posição " + i + " tem o componente: " + iterator.next());  
    i++;  
}
```

O iterator permite o acesso sequencial aos elementos de um ArrayList, facilitando a criação de relatórios para os sistemas.

Veja um exemplo prático com o uso de um ArrayList de Strings para um grupo de trabalho:

Classe: ArrayListTeste.

```
import java.util.ArrayList;  
import java.util.Iterator;  
import java.util.Scanner;  
public class ArrayListTeste {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        String antigo, novo;  
        int i, indice;  
        ArrayList<String> grupoTrabalho = new ArrayList<String>();  
        // para grupos de 5 componentes  
        for (i = 0; i < 5; i++) {  
            System.out.println("Digite o nome do " + (i + 1) + "º integrante");  
            grupoTrabalho.add(sc.nextLine());  
        }  
        System.out.println("Número de elementos do grupo: " + grupoTrabalho.size());  
        System.out.println("Substituição de um elemento por outro:");  
        System.out.println("Digite o nome do componente a ser substituído:");  
        antigo = sc.nextLine();  
        indice = grupoTrabalho.indexOf(antigo);  
        System.out.println("O Componente " + grupoTrabalho.get(indice) +  
            " Será substituído.");  
        System.out.println("Digite o nome do componente novo:");  
        novo = sc.nextLine();  
        grupoTrabalho.set(indice, novo);  
        System.out.println("Retirada de um elemento do grupo:");  
        System.out.println("Digite o nome do componente a ser eliminado:");  
        antigo = sc.nextLine();  
        indice = grupoTrabalho.indexOf(antigo);  
        grupoTrabalho.remove(indice);  
    }  
}
```

```

System.out.println("Grupo atual - Número de elementos do grupo: " +
    grupoTrabalho.size());
System.out.println("Elementos do grupo: ");
Iterator<String> iterator = grupoTrabalho.iterator();
i = 0;
// enquanto houver elementos no iterator, caso contrário sairá do laço
while (iterator.hasNext()) {
    // busca o próximo elemento no iterator
    System.out.println("Posição " + i + " tem o componente: " + iterator.next());
    i++;
}
System.out.println("Limpando o ArrayList:");
grupoTrabalho.clear();
System.out.println("Número de Elementos do grupo:" + grupoTrabalho.size());
}
}

```

Resultado da execução:

```

Digite o nome do 1º integrante
A
Digite o nome do 2º integrante
B
Digite o nome do 3º integrante
C
Digite o nome do 4º integrante
D
Digite o nome do 5º integrante
E
Número de elementos do grupo: 5
Substituição de um elemento por outro:
Digite o nome do componente a ser substituído:
C
O Componente C Será substituído.
Digite o nome do componente novo:
G
Retirada de um elemento do grupo:
Digite o nome do componente a ser eliminado:
B
Grupo atual - Número de elementos do grupo: 4
Elementos do grupo:
Posição 0 tem o componente: A
Posição 1 tem o componente: G
Posição 2 tem o componente: D
Posição 3 tem o componente: E
Limpando o ArrayList:
Número de Elementos do grupo:0

```

Acompanhe exemplos práticos de uso de vetores de objetos e uso de ArrayList de objetos para a classe Monitor:

Classe: Monitor - pacote: classes.

```

package classes;
import java.util.Scanner;
public class Monitor {

```



```
private String resolucao;
private double preco, potencia;
private int tamanhoTela, tensao;
public String getResolucao() {
    return resolucao;
}
public void setResolucao(String resolucao) {
    if (!resolucao.isEmpty()) {
        // são inseridos 15 espaços ao final para garantir que
        // tenha no mínimo 15 caracteres para o relatório
        this.resolucao = resolucao + " ";
    }
}
public double getPreco() {
    return preco;
}
public void setPreco(double preco) {
    if (preco >= 0) {
        this.preco = preco;
    }
}
public double getPotencia() {
    return potencia;
}
public void setPotencia(double potencia) {
    if (potencia >= 0) {
        this.potencia = potencia;
    }
}
public int getTamanhoTela() {
    return tamanhoTela;
}
public void setTamanhoTela(int tamanhoTela) {
    if (tamanhoTela >= 0) {
        this.tamanhoTela = tamanhoTela;
    }
}
public int getTensao() {
    return tensao;
}
public void setTensao(int tensao) {
    if (tensao >= 0) {
        this.tensao = tensao;
    }
}
public Monitor() { }
public Monitor(String resolucao, double preco, double potencia, int tamanhoTela,
               int tensao) {
    setResolucao( resolucao );
    setPreco( preco );
    setPotencia( potencia );
    setTamanhoTela( tamanhoTela );
    setTensao( tensao);
}
public Monitor(String resolucao, double preco, int tamanhoTela) {
    setResolucao( resolucao );
    setPreco( preco );
    setTamanhoTela( tamanhoTela );
}
public void cadastrar(String resolucao, double preco, double potencia,
```

```

        int tamanhoTela, int tensao) {
    setResolucao( resolucao );
    setPreco( preco );
    setPotencia( potencia );
    setTamanhoTela( tamanhoTela );
    setTensao( tensao);
}

public void imprimir() {
    System.out.println("Resolução : " + resolucao);
    System.out.println("Preço : " + preco);
    System.out.println("Potência : " + potencia);
    System.out.println("Tamanho : " + tamanhoTela);
    System.out.println("Tensão : " + tensao);
}

public void entradaDados() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Resolução : ");
    setResolucao( sc.nextLine() );
    System.out.println("Preço : ");
    setPreco( Double.parseDouble(sc.nextLine() ));
    System.out.println("Potência : ");
    potencia = Double.parseDouble(sc.nextLine());
    System.out.println("Tamanho : ");
    setTamanhoTela( Integer.parseInt(sc.nextLine() ));
    System.out.println("Tensão : ");
    setTensao( Integer.parseInt(sc.nextLine() ));
}
}

```

● ● ● Classe: VetorObjetos - pacote: aplicacoes.

```

package aplicacoes;
import java.util.Scanner;
import classes.*;
public class VetorObjetos {
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner ent = new Scanner(System.in);
        // determinação da quantidade de elementos
        System.out.println("Quantos monitores serão inseridos?");
        int qtd = Integer.parseInt(ent.nextLine());
        Monitor monit[] = new Monitor[qtd]; // declaração
        for(int i=0; i
            monit[i] = new Monitor(); // Instanciação de cada objeto
            monit[i].entradaDados();
        }
        for(int i=0; i<monit.length; i++){
            System.out.println("*****");
            monit[i].imprimir();
            System.out.println("*****");
        }
    }
}

```

● ● ● Classe: ArrayListObjetos - pacote: aplicacoes.

```

package aplicacoes;

```

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
import classes.Monitor;
public class ArrayListObjetos {
public static void main(String[] args) {
    // TODO code application logic here
    boolean continua = true;
    Scanner ent = new Scanner(System.in);
    ArrayList<Monitor> listaMonitores = new ArrayList<Monitor>();
    do {
        System.out.println("Inserir Monitor [1-Sim, 2-Não]?");
        if (Integer.parseInt(ent.nextLine()) == 1) {
            // Inserir um objeto monitor
            // objeto auxiliar para entrada de dados e inserção a lista
            Monitor aux = new Monitor();
            aux.entradaDados(); // entrada de dados do objeto
            listaMonitores.add(aux); // inserção a lista
        } else {
            // Não inserir um objeto monitor
            // encerrar a entrada de dados
            continua = false;
        }
    } while (continua);

    // criação do objeto iterator para o acesso sequencial e criação do relatório
    Iterator<Monitor> iterator = listaMonitores.iterator();
    // criação do cabeçalho do relatório
    System.out.println("");
    System.out.println(": Resolução : Preço : Potência :");
    // contador de elementos
    int cont = 0;
    // enquanto houver elementos no iterator, caso contrário sairá do laço
    while (iterator.hasNext()) {
        // objeto auxiliar para entrada de dados e inserção a lista
        Monitor aux = new Monitor();
        // busca o próximo elemento no iterator e transfere os dados para o
        // objeto auxiliar
        aux = (Monitor) iterator.next();
        System.out.println("");
        System.out.printf("%-15s : %8.2f : %8.2f" ,
            aux.getResolucao().substring(0, 14),
            aux.getPreco(),
            aux.getPotencia());

        cont++;
    }
    // criação do rodapé do relatório
    System.out.println("");
    System.out.println("Quantidade de Monitores: " + cont);
}
}

```

Resultado da execução:

```

Inserir Monitor [1-Sim, 2-Não]?
1
Resolução :
200
Preço :

```

```
200
Potência :
200
Tamanho :
200
Tensão :
200
Inserir Monitor [1-Sim, 2-Não]?
1
Resolução :
2000
Preço :
2000
Potência :
2000
Tamanho :
2000
Tensão :
2000
Inserir Monitor [1-Sim, 2-Não]?
1
Resolução :
20000
Preço :
20000
Potência :
20000
Tamanho :
20000
Tensão :
20000
Inserir Monitor [1-Sim, 2-Não]?
2
: Resolução      : Preço           : Potência :
200               : 200,00          : 200,00
2000              : 2000,00         : 2000,00
20000             : 20000,00        : 20000,00
Quantidade de Monitores: 3
```

Notas

- 1

Foi criada a classe Monitor para uso das aplicações;
- 2

A aplicação VetorObjetos trabalha com a definição de um vetor de objetos Monitor, de acordo com a quantidade de elementos determinados pelo usuário;
- 3

A aplicação ArrayListObjetos trabalha com o uso de um *ArrayList* de Monitores, sendo incluídos elementos de acordo com a necessidade do usuário;
- 4

Na classe ArrayListObjetos foi trabalhado o uso convencional de um ArrayList, em que cada elemento é tratado como um objeto à parte. Nos exemplos dados, foram inseridos a partir de uma entrada de dados. Entretanto, os dados poderiam ter sido adquiridos pelo resultado de uma consulta a uma base de dados, e esses registros inseridos no ArrayList;

O uso de *ArrayList* é muito importante ao desenvolvermos sistemas e trabalharmos com dados armazenados em tabelas de banco de dados.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

I Atividade

1) Sobre tratamento de dados homogêneos, é correto afirmar que:

- I. Para declarar um vetor, obrigatoriamente temos que definir o seu tamanho.
- II. Ao acessar uma área fora do intervalo do vetor, ocorrerá uma exceção (ArrayIndexOutOfBoundsException), mas a execução do programa continuará normalmente.
- III. Para criar vetores multidimensionais, basta usar mais de uma dimensão na definição e na criação do vetor.

Assinale a alternativa correta:

- ☐ a) Somente a afirmativa I é verdadeira.
- ☐ b) Somente a afirmativa II é verdadeira.
- ☐ c) Somente a afirmativa III é verdadeira.
- ☐ d) Somente as afirmativas I e II são verdadeiras.
- ☐ e) Somente as afirmativas II e III são verdadeiras.

Notas

abruptamente¹

Término causado por algum erro.

Referências

DEITEL, Paul. **Java: como programar** (Biblioteca Virtual). 10a ed. São Paulo: Pearson, 2017.

Explore mais

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.