

# Domain Driven Design

## Eliane Marion

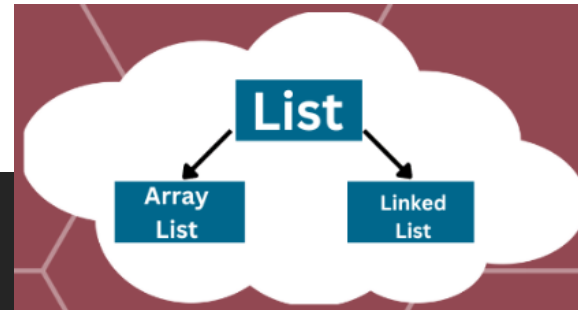
FIAP

2024

# AGENDA DE TRABALHO



VETORES



LISTAS

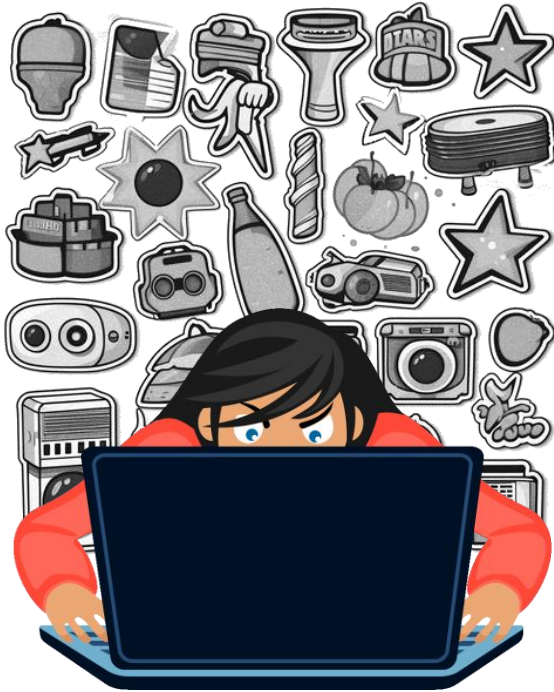


# 01

Vetores

Revisão

# Vetores



- São estruturas de dados que permitem armazenar vários dados simultaneamente.
- Os dados sempre serão de mesmo tipo.
- O seu tamanho deve ser definido no momento da sua criação na memória

# EXEMPLO


```
public class Vetores {  
    public static void main(String[] args) {  
        Scanner entrada = new Scanner(System.in);  
        //Declaração de um vetor de strings chamado nome  
        String nome = new String[];  
        // Leitura do vetor acima  
        for(int i = 0; i <= 4; i++){  
            System.out.println("Digite o" + (i+1) + " nome:");  
            nome[i]= leitor.nextLine();  
  
        nome[i]= leitor.nextLine();  
    }  
  
        n3 = entrada.nextFloat();  
        n4 = entrada.nextFloat();  
        media = (n1 + n2 + n3 + n4) / 4;  
  
        System.out.println("Média = " + media);  
    }  
}
```

# EXEMPLO

```
public class Vetores {  
    public static void main(String[] args) {  
        Scanner entrada new Scanner(System.in);  
  
        //Declaração de um vetor de strings chamado nome  
        String nome = new String[];
```



# EXEMPLO

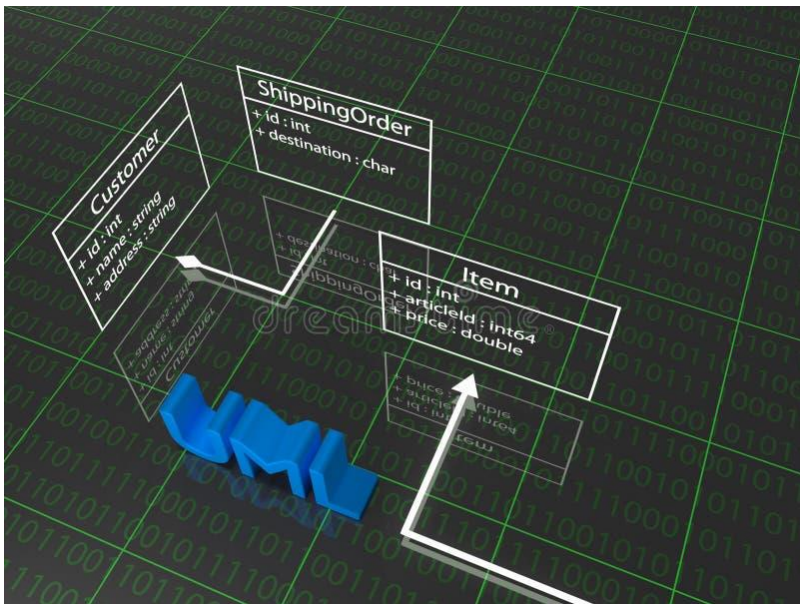
```
public class Vetores {  
    public static void main(String[] args) {  
        Scanner entrada new Scanner(System.in);  
        //Declaração de um vetor de strings chamado nome  
        String nome = new String[];  
        // Leitura do vetor acima  
        for(int i = 0; i <= 4; i++){  
            stem.out.println("Digite o" + (i+1) + " nome:");  
            nome[i] = leitor.nextLine();  
        }  
    }  
}
```

Eliane

i = 0

Digite o 1 nome:

Eliane

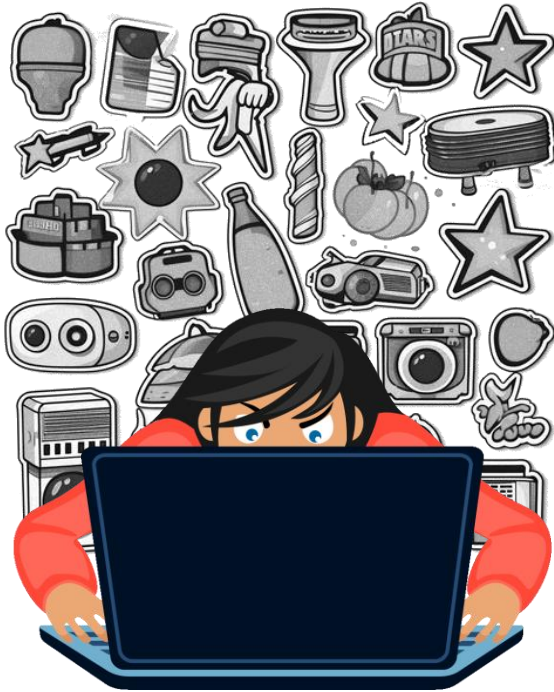


# 02

## Collections Framework

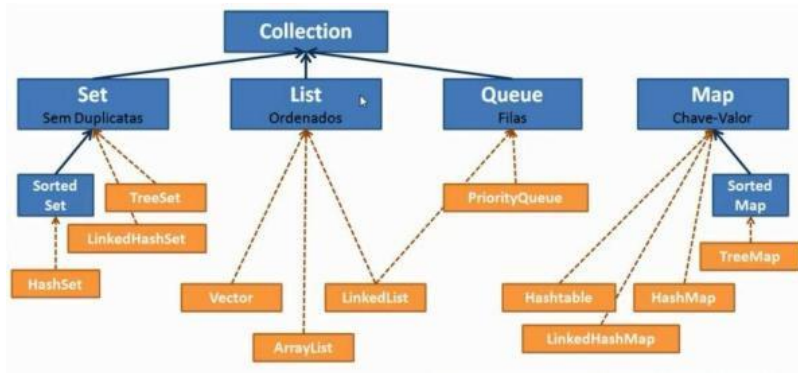


# Collections Framework



É um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade, que pode ser chamada coleção, ou collection. A Collections Framework contém os seguintes elementos: Interfaces, Implementações, Algoritmos.:

# Collections Framework

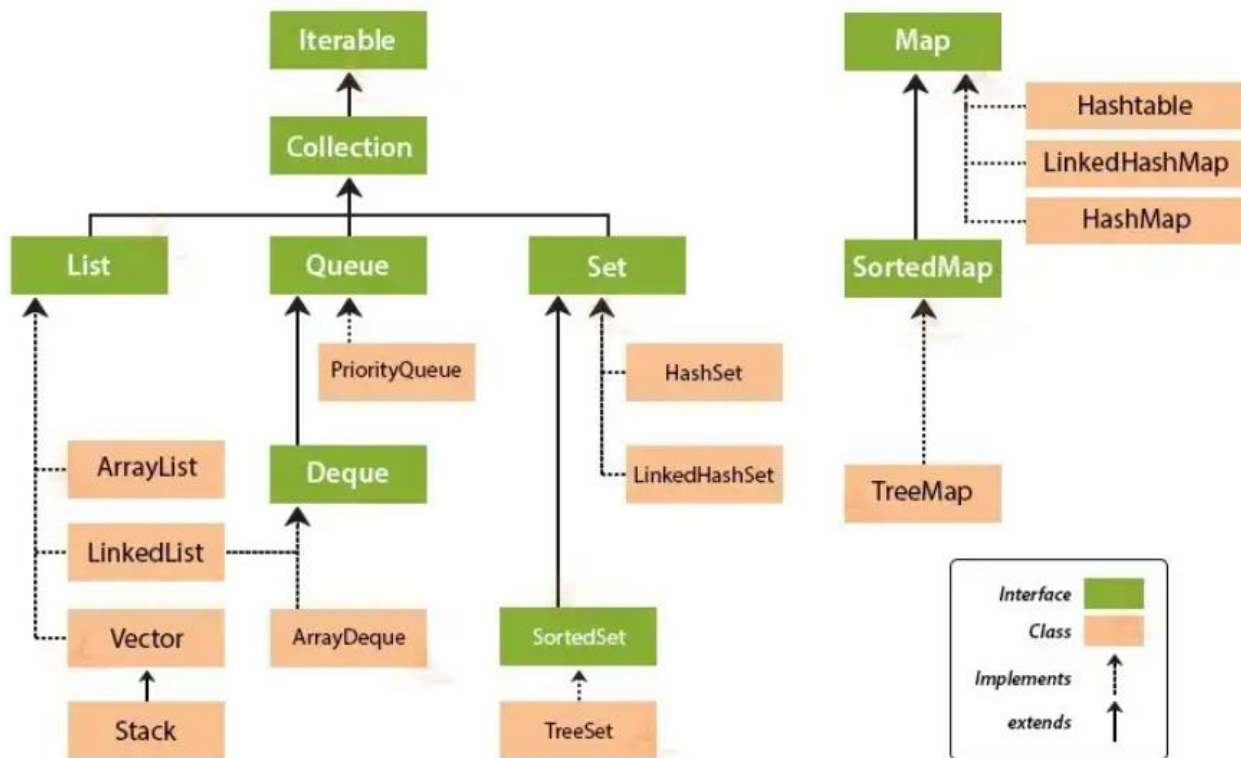


**Interfaces:** tipos abstratos que representam as coleções. Permitem que coleções sejam manipuladas tendo como base o conceito”, desde que o acesso aos objetos se restrinja apenas ao uso de métodos definidos nas interfaces;

**Implementações:** são as implementações concretas das interfaces;

**Algoritmos:** são os métodos que realizam as operações sobre os objetos das coleções

# UML E O DIAGRAMA DE CLASSES



# Java.util.Collection

---

## Interface Iterable

Modificador e tipo	Método e descrição
default void	<code>foreach(Consumer&lt;? super T&gt; action)</code> Executa a ação especificada para cada elemento do Iterable até que todos tenham sido processados.
<code>Iterator&lt;T&gt;</code>	<code>Iterator&lt;T&gt;</code> Retorna um iterator sobre elementos do tipo T

Tem como função principal permitir que qualquer coleção derivada seja iterável pelo laço `foreach` ;

Permite tipagem, Isto é feito com o uso de Generics e garante um melhor controle da coleção, pois permite que sejam notados erros de programação ainda em tempo de compilação.

# Java.util.Collection

---

## Interface Collection

Não existe implementação direta dessa interface, mas ela define as operações básicas para as coleções, como adicionar, remover, esvaziar, etc.

Está na raiz das interfaces do primeiro grupo da **Collections Framework**.

Devem ser objetos.

Define os métodos comuns entre as suas subinterfaces, padronizando as operações básicas disponíveis para essas coleções;

# Implementações

---

**Set** – interface que define uma coleção que não permite elementos duplicados. A interface SortedSet, que estende Set, possibilita a classificação natural dos elementos, tal como a ordem alfabética;

**Queue** – um tipo de coleção para manter uma lista de prioridades, onde a ordem dos seus elementos, definida pela implementação de Comparable ou Comparator, determina essa prioridade. Com a interface fila pode-se criar filas e pilhas;

**List** – define uma coleção ordenada, podendo conter elementos duplicados. Em geral, o usuário tem controle total sobre a posição onde cada elemento é inserido e pode recuperá-los através de seus índices. Prefira esta interface quando precisar de acesso aleatório, através do índice do elemento;

# Implementações da Interface **List**



**ArrayList** – é como um array cujo tamanho pode aumentar. A busca de um elemento é rápida, mas inserções e exclusões não são. Podemos afirmar que as inserções e exclusões são lineares, o tempo cresce com o aumento do tamanho da estrutura. Esta implementação é preferível quando se deseja acesso mais rápido aos elementos.

**LinkedList** – implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões

# EXEMPLO DA UTILIZAÇÃO DE LIST

---

```
//Declarando um ArrayList
ArrayList<Filme> listaDeFilmes = new ArrayList<>();

//Adicionando um filme à lista
listaDeFilmes.add(meuFilme);
listaDeFilmes.add(outroFilme);

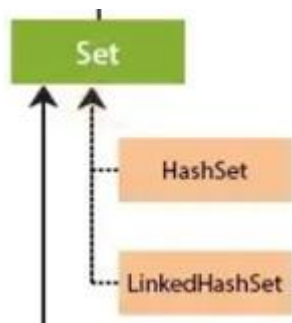
//Exibindo a quantidade de elementos da lista
System.out.println("Você assistiu " + listaDeFilmes.size());

//Exibindo o nome do primeiro elemento da lista
System.out.println("O primeiro filme que você assistiu foi " +
    listaDeFilmes.get(0).getNome());

//Percorrendo a lista
for(Filme filme : listaDeFilmes){
    System.out.println(filme.getNome());
}
```



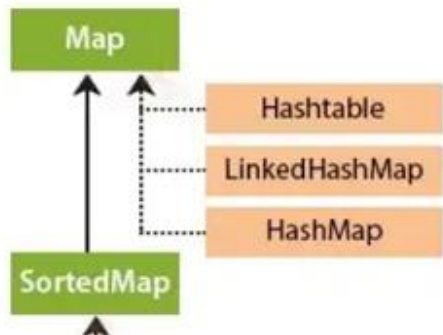
# Implementações da Interface **Set**



**HashSet** – o acesso aos dados é mais rápido que em um **TreeSet**, mas nada garante que os dados estarão ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante.

**LinkedHashSet** – é derivada de **HashSet**, mas mantém uma lista duplamente ligada através de seus itens. Seus elementos são iterados na ordem em que foram inseridos. Opcionalmente é possível criar um **LinkedHashSet** que seja percorrido na ordem em que os elementos foram acessados na última iteração.

# Interfaces **Map** e **SortedMap**

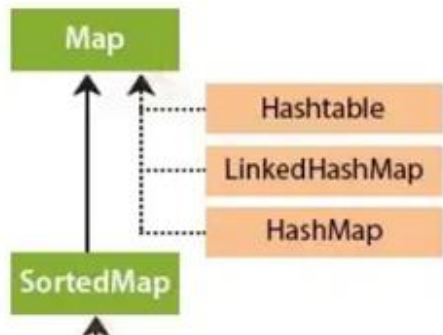


**SortedMap** - é uma interface que estende Map, e permite classificação ascendente das chaves.

Uma aplicação dessa interface é a classe Properties, que é usada para persistir propriedades/configurações de um sistema, por exemplo.

**Map** – mapeia chaves para valores. Cada elemento tem na verdade dois objetos: uma chave e um valor. Valores podem ser duplicados, mas chaves não.

# Implementações da Interface **Map** e **SortedMap**



**LinkedHashMap** – mantém uma lista duplamente ligada através de seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa. Se for necessário um mapa onde os elementos são iterados na ordem em que foram inseridos, use esta implementação.

**HashMap** – baseada em tabela de espalhamento, permite chaves e valores null. Não existe garantia que os dados ficarão ordenados. Escolha esta implementação se a ordenação não for importante e desejar uma estrutura onde seja necessário um ID (identificador).

# Estrutura básica da Interface **Map**

A interface Map<K, V> tem dois parâmetros genéricos:

k -> Tipo das chaves | V -> Tipo dos valores

Métodos	Descrição
put(K key, V value)	Adiciona um par chave-valor ao mapa.
get(Object key)	Retorna o valor associado à chave fornecida.
remove(Object key)	Remove o par chave-valor para a chave especificada.
containsKey(Object key)	Verifica se o mapa contém a chave especificada.
containsValue(Object value)	Verifica se o mapa contém o valor especificado.
size()	Retorna o número de pares chave-valor no mapa.
isEmpty()	Verifica se o mapa está vazio.
keySet()	Retorna um Set com todas as chaves no mapa.
values()	Retorna uma Collection com todos os valores no mapa.

# Principais implementações: **HashMap**

```
• //Declarando um HashMap
• Map<String, Integer> map = new HashMap<>();
•

//Adicionando itens
map.put("Eliane", 45);
map.put("Ana", 18);
map.put("Teste", 30);

//Exibindo a idade da Ana
System.out.println("Idade da Ana" + map.get("Ana"));

//Excluindo um elemento
map.remove("Teste");

System.out.println("Contem Teste? " + map.containsKey("Teste"));
```

# Novidades

---

A partir do **Java 8**, foi adicionado na interface `List`, a qual a classe `ArrayList` implementa, um novo método chamado **`forEach`**, que possibilita a iteração sobre os elementos da lista de forma mais concisa e elegante.

O método **`forEach`** é chamado sobre a lista `nomes` e recebe como parâmetro uma expressão lambda que realiza a impressão do valor na tela.

A expressão **lambda** `listaDeFilmes -> System.out.println(filme.getNome())` é uma forma compacta de definir uma função que recebe um parâmetro `filme` e realiza a operação de impressão.

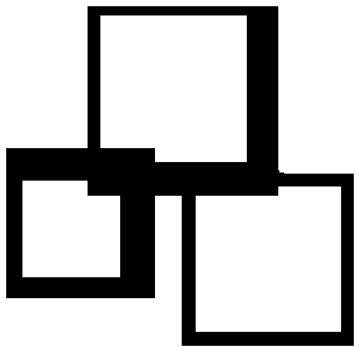
```
listaDeFilmes.forEach(  
    filme -> System.out.println(filme.getNome())  
);
```

# Novidades

---

```
nomes.forEach(System.out::println);
```

É possível simplificar ainda mais o exemplo de código anterior, utilizando o recurso conhecido como Method Reference, que nada mais é do que uma forma reduzida de uma expressão lambda:



**OBRIGADO**

*To be continued...*

BYE  
BYE