Programação III

Aula 5 – Collections

Definição

- Collections são estruturas de dados flexíveis que armazenam e manipulam conjuntos de elementos de forma eficiente.
- Elas fazem parte do pacote **java.util** e oferecem funcionalidades avançadas para gerenciar listas, conjuntos, mapas e filas.

Característica	Arrays	Collections
Tamanho	Fixo	Dinâmico
Tipo de Dados	Homogêneo	Pode armazenar objetos genéricos (<T>)
Métodos	Limitados (acesso por índice, atribuição)	Métodos avançados (busca, ordenação, filtragem, remoção, etc.)
Eficiência	Melhor para acesso direto por índice	Melhor para manipulação dinâmica de dados

Vantagens do uso

✅ **Flexibilidade** – Permite trabalhar com diferentes estruturas de dados sem precisar gerenciar manualmente alocação e redimensionamento.

✅ **Métodos Utilitários** – Inclui métodos para ordenação (`sort`), busca (`binarySearch`), e manipulação de listas, conjuntos e mapas.

✅ **Eficiência** – Implementações otimizadas para melhor desempenho de acesso, inserção e remoção.

✅ **Consistência** – Todas as implementações seguem padrões unificados, tornando o código mais legível e modular.

Estrutura das Collections

- A interface base do framework é `Collection<E>`, da qual derivam três principais subinterfaces:
  - 📌 **`List<E>`** – Estruturas ordenadas que permitem elementos duplicados.
  - 📌 **`Set<E>`** – Estruturas **sem duplicatas**, que podem ou não manter a ordem.
  - 📌 **`Queue<E>`** – Estruturas que seguem um modelo de **fila (FIFO – First In, First Out).**

  Além disso, existe a interface **`Map<K, V>`**, que representa uma coleção de pares **chave-valor** e **não estende `Collection` diretamente**.

Principais Implementações

- Cada interface possui diferentes classes concretas para atender a diferentes necessidades:

Interface	Implementações Principais	Características
List	`ArrayList, LinkedList`	Ordenadas, permitem duplicatas
Set	`HashSet, TreeSet, LinkedHashSet`	Sem duplicatas, pode ter ordenação
Queue	`LinkedList, PriorityQueue`	Fila (FIFO) ou prioridade
Map	`HashMap, TreeMap, LinkedHashMap`	Estruturas de chave-valor

List

- Estrutura de lista ordenada, permite duplicatas.
- Principais implementações:
  - `ArrayList` (rápido para leitura, acesso por índice).
  - `LinkedList` (eficiente para inserções e remoções no meio).

```java
List<String> lista = new ArrayList<>();
lista.add("Java");
lista.add("Python");
lista.remove("Python");
System.out.println(lista.get(0)); // Java
```

Set

- Conjunto de elementos únicos.
- Principais implementações:
  - `HashSet` (sem ordem específica).
  - `LinkedHashSet` (mantém ordem de inserção).
  - `TreeSet` (elementos ordenados).

```java
Set<String> set = new HashSet<>();
set.add("Maçã");
set.add("Banana");
set.add("Maçã"); // Ignorado
System.out.println(set); //[Maçã, Banana]
```

Map

- Estrutura **chave-valor**.
- Implementações principais:
  - `HashMap` (rápido, sem ordem).
  - `LinkedHashMap` (ordem de inserção).
  - `TreeMap` (chaves ordenadas).

```java
Map<Integer, String> mapa = new HashMap<>();
mapa.put(1, "João");
mapa.put(2, "Maria");
System.out.println(mapa.get(2)); // Maria
```

Queue

- Estrutura de fila (FIFO – First In, First Out).
- Implementações principais:
  - `LinkedList` (como fila).
  - `PriorityQueue` (ordenação por prioridade).

```java
Queue<String> fila = new LinkedList<>();
fila.add("A");
fila.add("B");
System.out.println(fila.poll()); // Remove "A"
```

Métodos Utilitários

- **Ordenação:** `Collections.sort(lista);`
- **Busca binária:** `Collections.binarySearch(lista, elemento);`
- **Embaralhamento:** `Collections.shuffle(lista);`

```
List<Integer> numeros = Arrays.asList(5, 1, 8, 3);
Collections.sort(numeros);
System.out.println(numeros); // [1, 3, 5, 8]
```