UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

AULA 07

LINGUAGEM DE PROGRAMAÇÃO 2
JAVA

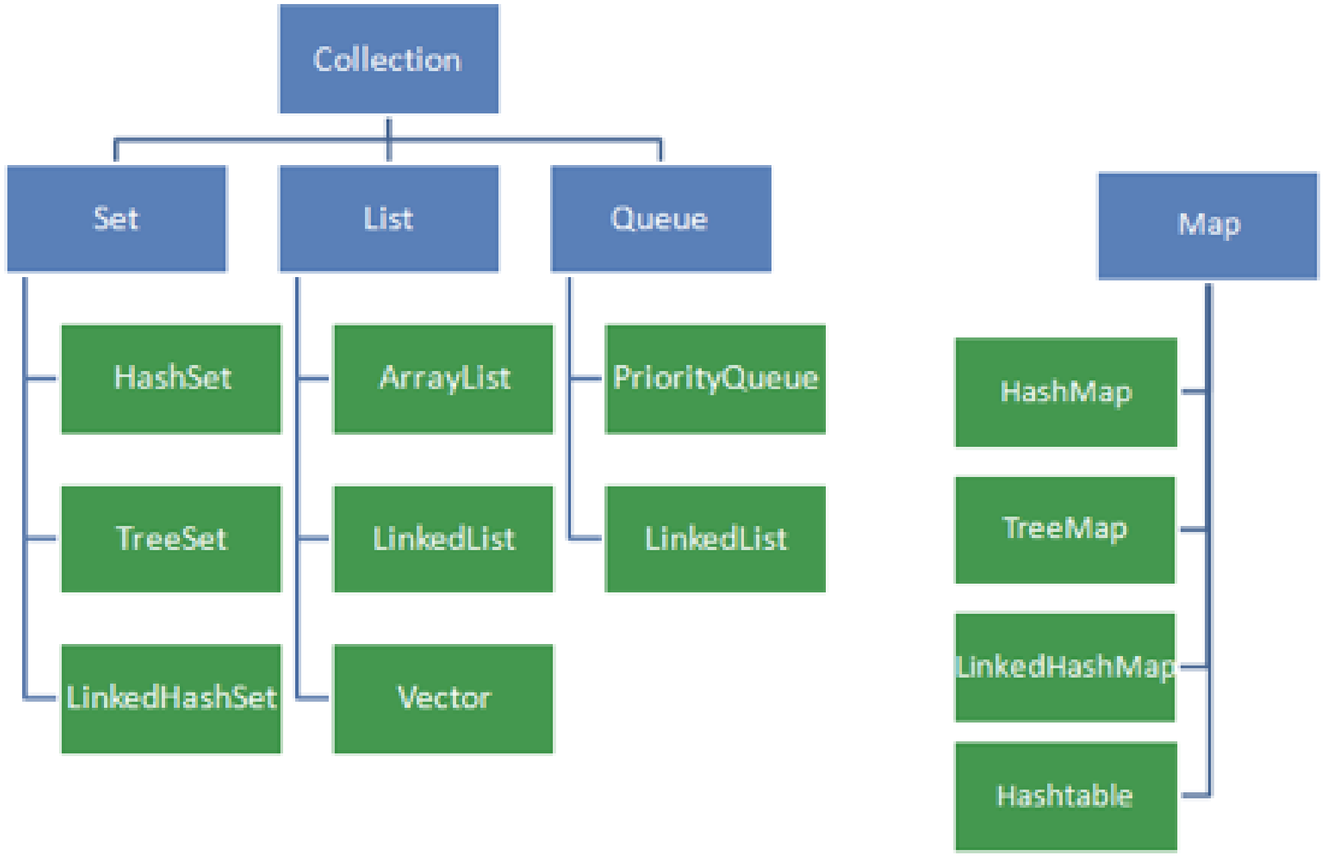
PROF. JANIHERYSON FELIPE

CONTEÚDO DESSA AULA

- CONHECER OS CONCEITOS DE COLLECTIONS NO JAVA;
- CONHECER AS PRINCIPAIS COLLECTIONS E SEUS RESPECTIVOS MÉTODOS;
- DISCUSSÕES E DÚVIDAS GERAIS.

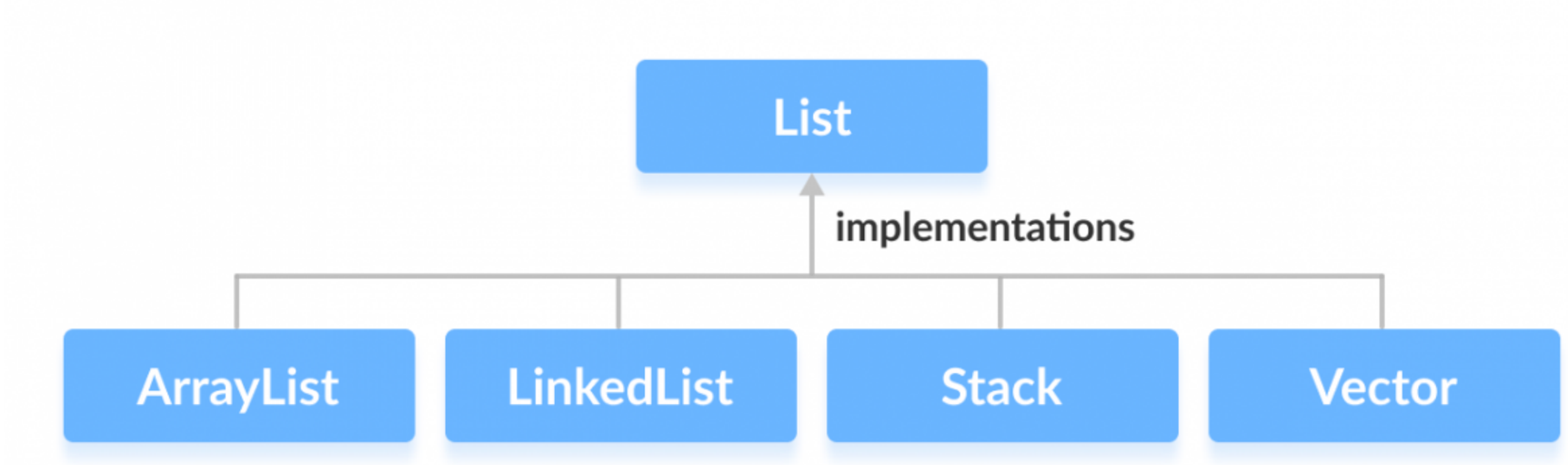
COLLECTIONS EM JAVA

- Em Java, o pacote java.util fornece uma série de estruturas de dados conhecidas como "collections" (coleções), que são projetadas para armazenar, manipular e organizar grupos de objetos de maneira eficiente.



LIST

List: Uma coleção ordenada que permite elementos duplicados. As listas permitem acessar elementos pelo seu índice e também oferecem operações para adicionar, remover e modificar elementos. Exemplos de implementações de List incluem **ArrayList**, **LinkedList** e **Vector**.



ARRAYLIST

O ArrayList é uma implementação de listas em Java que armazena elementos **dinamicamente** e permitem acessá-los por meio de um **índice**.

- **add(E element)**: Adiciona um elemento à lista.
- **remove(int index)**: Remove o elemento na posição especificada na lista.
- **get(int index)**: Retorna o elemento na posição especificada na lista.
- **set(int index, E element)**: Substitui o elemento na posição especificada
- **size()**:Retorna o número de elementos na lista.
- **isEmpty()**:Retorna verdadeiro se a lista estiver vazia, falso caso contrário.
- **clear()**:Remove todos os elementos da lista.

ARRAYLIST

- **contains(Object o):** Retorna verdadeiro se a lista contiver o elemento especificado.
- **indexOf(Object o):** Retorna o índice da primeira ocorrência do elemento especificado na lista, ou -1 se a lista não contiver o elemento.
- **lastIndexOf(Object o):** Retorna o índice da última ocorrência do elemento especificado na lista, ou -1 se a lista não contiver o elemento.

```
import java.util.ArrayList;
```

ARRAYLIST

```java
ArrayList<String> list = new ArrayList<>();
list.add("Elemento1");
list.add("Elemento2");
list.add("Elemento3");
list.add("Elemento4");

for(int i = 0; i < list.size(); i++){
    System.out.println(list.get(i));
}
for(String element : list){
    System.out.println(element);
}
```

VECTOR

O Vector é uma implementação de listas em Java que armazena elementos **dinamicamente** e permitem acessá-los por meio de um **índice**.

- **add(E element)**: Adiciona um elemento à lista.
- **remove(int index)**: Remove o elemento na posição especificada na lista.
- **get(int index)**: Retorna o elemento na posição especificada na lista.
- **set(int index, E element)**: Substitui o elemento na posição especificada
- **size()**:Retorna o número de elementos na lista.
- **isEmpty()**:Retorna verdadeiro se a lista estiver vazia, falso caso contrário.
- **clear()**:Remove todos os elementos da lista.

VECTOR

- **contains(Object o):** Retorna verdadeiro se a lista contiver o elemento especificado.
- **indexOf(Object o):** Retorna o índice da primeira ocorrência do elemento especificado na lista, ou -1 se a lista não contiver o elemento.
- **lastIndexOf(Object o):** Retorna o índice da última ocorrência do elemento especificado na lista, ou -1 se a lista não contiver o elemento.

```
import java.util.Vector;
```

VECTOR

```java
Vector<String> vector = new Vector<>();
vector.add("Elemento1");
vector.add("Elemento2");
vector.add("Elemento3");
vector.add("Elemento4");

for(int i = 0; i < vector.size(); i++){
    System.out.println(vector.elementAt(i));
}
for(String element : vector){
    System.out.println(element);
}
```

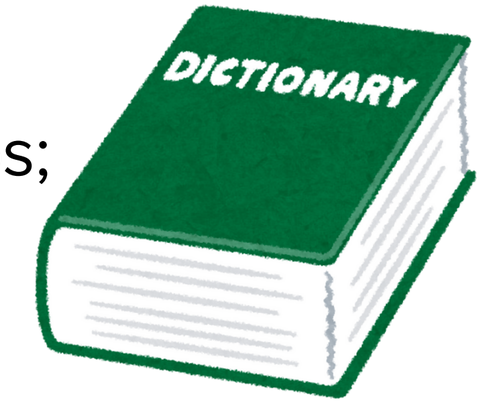
ARRAYLIST VS VECTOR

- Vector é **sincronizado** por padrão, o que significa que é **seguro** para uso em **ambientes multithreading.** Todas as operações do Vector são sincronizadas, o que garante que duas ou mais **threads não possam modificar o Vector ao mesmo tempo**. ArrayList, por outro lado, não é sincronizado. Isso significa que **não é seguro** para uso em ambientes **multithreading**;
- **ArrayList** tende a ter **melhor desempenho** do que Vector, principalmente em cenários não concorrentes. Isso ocorre porque ArrayList não tem o custo adicional da sincronização em cada operação.

MAP

Representa uma coleção de pares **chave-valor**, onde cada **chave é única** e mapeia para um único valor. O pacote **java.util** fornece várias implementações da interface Map, cada uma com suas próprias características e comportamentos. (HashMap, LinkedHashMap, TreeMap)

- HashMap: Não se importa com a ordem dos elementos;
- LinkedHashMap: Manter a ordem de inserção dos elementos;
- TreeMap: Ordena todos os elementos pela chave;



MAP - MÉTODOS

put(key, value): Adiciona um par chave-valor ao mapa. Se a chave já existir, o valor associado à chave será substituído pelo novo valor e o valor anterior será retornado.

get(key): Retorna o valor associado à chave especificada no mapa, ou null se a chave não estiver presente.

remove(key): Remove a entrada correspondente à chave especificada do mapa, se ela existir. Retorna o valor associado à chave removida, ou null se a chave não estiver presente.

MAP - MÉTODOS

containsKey(key): Verifica se o mapa contém a chave especificada. Retorna true se a chave estiver presente, caso contrário, retorna false.

containsValue(value): Verifica se o mapa contém o valor especificado. Retorna true se o valor estiver presente, caso contrário, retorna false.

size(): Retorna o número de pares chave-valor no mapa.]

isEmpty(): Retorna true se o mapa estiver vazio, caso contrário, retorna false.

clear(): Remove todos os pares chave-valor do mapa.

MAP - MÉTODOS

keySet(): Retorna um conjunto contendo todas as chaves no mapa. Esse conjunto pode ser iterado para obter as chaves individuais.

values(): Retorna uma coleção contendo todos os valores no mapa. Essa coleção pode ser iterada para obter os valores individuais.

entrySet(): Retorna um conjunto de entradas (pares chave-valor) no mapa. Cada entrada é representada por um objeto Map.Entry<K, V>, que possui métodos getKey() e getValue() para recuperar a chave e o valor, respectivamente.

MAP - MÉTODOS

```java
import java.util.HashMap;
```

```java
HashMap<String, Integer> map = new HashMap<>();


map.put("A", 1);
map.put("B", 2);
map.put("C", 3);
map.put("D", 4);
map.put("E", 5);
```

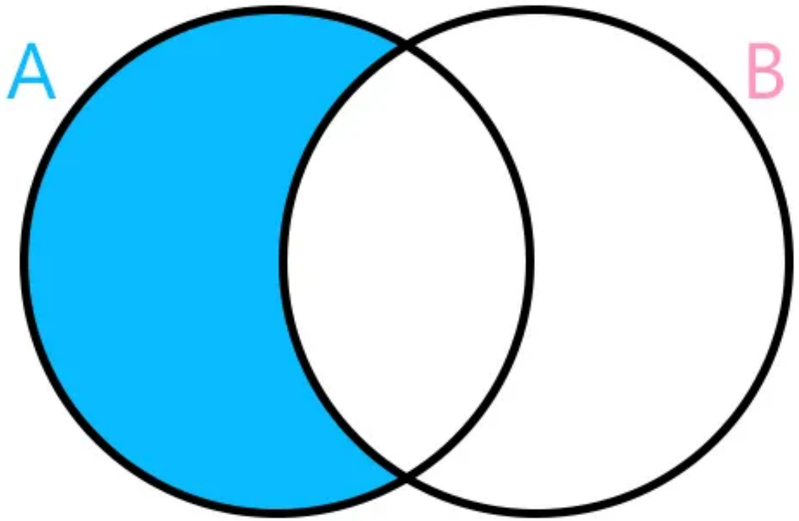
ITERANDO SOBRE MAPAS

```java
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    String key = entry.getKey();
    Integer value = entry.getValue();
    System.out.println("Chave: " + key + ", Valor: " + value);
}

map.forEach((key, value) -> {
    System.out.println("Chave: " + key + ", Valor: " + value);
});
```

SETS - CONJUNTOS

Representa uma coleção que não permite elementos duplicados. Isso significa que cada elemento em um conjunto deve ser único. O pacote java.util fornece várias implementações da interface Set, cada uma com suas próprias características e comportamentos.



SETS - MÉTODOS

add(): Adiciona o elemento especificado ao conjunto, se ele ainda não estiver presente.

remove(): Remove o elemento especificado do conjunto, se estiver presente.

contains(): Verifica se o conjunto contém o elemento especificado. Retorna true se o elemento estiver presente, caso contrário, retorna false.

isEmpty(): Retorna true se o conjunto estiver vazio, caso contrário, retorna false.

size(): Retorna o número de elementos no conjunto.

clear(): Remove todos os elementos do conjunto.

SETS - IMPLEMENTAÇÃO

```
Set<String> set = new HashSet<>();


set.add("A");
set.add("B");
set.add("C");
set.add("A"); // Tentativa de adicionar um elemento duplicado
set.add("D");
set.add("B"); // Tentativa de adicionar outro elemento duplicado
```

SETS - IMPLEMENTAÇÃO

```java
import java.util.Set;
import java.util.HashSet;
```

```java
System.out.println("Elementos do conjunto:");
for (String element : set) {
    System.out.println(element);
}
```