

# Estruturas de Dados

- Listas (List)

- Sequências com noção de ordem, com repetição
- Podem conter duplicados

```
public interface List<E> extends Collection<E> {  
    // Positional Access  
    boolean add(E e);  
    void add(int index, E element);           // Optional  
    E get(int index);  
    E set(int index, E element);             // Optional  
    E remove(int index);                    // Optional  
    boolean addAll(Collection<? extends E> c); // Optional  
  
    // Search  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
  
    // Iteration  
    ListIterator<E> listIterator();  
    ListIterator<E> listIterator(int index);  
  
    // Range-view  
    List<E> subList(int from, int to);  
}  
...  
public interface ListIterator<E>  
    extends Iterator<E> {  
    boolean hasNext();  
    E next();  
    boolean hasPrevious();  
    E previous();  
    int nextIndex();  
    int previousIndex();  
    void remove(); //optional  
    void set(E e); //optional  
    void add(E e); //optional  
}
```

- Mais comuns:
  - ArrayList (Array Dinâmico)
  - LinkedList (Lista ligadas)

## ArrayList x LinkedList

A **LinkedList** classe é uma coleção que pode conter muitos objetos do mesmo tipo, assim como o arquivo **ArrayList**.

A **LinkedList** classe possui todos os mesmos métodos da **ArrayList** classe porque ambos implementam a **List** interface. Isso significa que você pode adicionar itens, alterar itens, remover itens e limpar a lista da mesma maneira.

No entanto, embora a `ArrayList` classe e a `LinkedList` classe possam ser usadas da mesma maneira, elas são construídas de maneira muito diferente.

## Como funciona o ArrayList

A `ArrayList` classe possui um array regular dentro dela. Quando um elemento é adicionado, ele é colocado no array. Se o array não for grande o suficiente, um novo array maior será criado para substituir o antigo e o antigo será removido.

## Como funciona o LinkedList

A `LinkedList` armazena seus itens em "contêineres". A lista possui um link para o primeiro contêiner e cada contêiner possui um link para o próximo contêiner da lista. Para adicionar um elemento à lista, o elemento é colocado em um novo contêiner e esse contêiner é vinculado a um dos outros contêineres da lista.

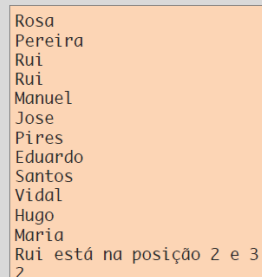
## Quando usar

Use um `ArrayList` para armazenar e acessar dados e `LinkedList` para manipular dados.

- Outras:
  - Vector (Array Dinâmico)
  - Stack

Exemplo:

```
public static void main(String args[]) {  
    String[] str1 = {"Rui", "Manuel", "Jose", "Pires", "Eduardo", "Santos"};  
    String[] str2 = {"Rosa", "Pereira", "Rui", "Vidal", "Hugo", "Maria"};  
    List<String> larray = new ArrayList<>();  
    List<String> llist = new LinkedList<>();  
  
    for (String i: str1 ) larray.add(i);  
    for (String i: str2 ) llist.add(i);  
  
    llist.addAll(llist.size()/2, larray);  
    for (String ele: llist)  
        System.out.println( ele );  
  
    System.out.println("Rui está na posição " +  
        llist.indexOf("Rui") + " e " + llist.lastIndexOf("Rui"));  
  
    llist.set(llist.lastIndexOf("Rui"), "Rui2");  
    System.out.println(llist.lastIndexOf("Rui"));  
}
```



```
Rosa  
Pereira  
Rui  
Rui  
Manuel  
Jose  
Pires  
Eduardo  
Santos  
Vidal  
Hugo  
Maria  
Rui está na posição 2 e 3  
2
```

- Filas (Queue)

- São as filas do tipo *First in First Out*

Exemplo:

```
public interface Queue<E> extends Collection<E> {  
  
    // Inserts the specified element in the queue  
    boolean offer(E e);  
  
    // Retrieves and removes the head of this queue  
    // throws an exception if empty  
    E remove();  
    // Retrieves and removes the head of this queue  
    E poll();  
  
    // Retrieves, but does not remove, the head of this queue  
    // throws an exception if empty  
    E element();  
    // Retrieves, but does not remove, the head of this queue  
    E peek();  
}
```



- Conjuntos (Set)

- Sem noção de posição (sem ordem), com repetição
  - Não pode conter elementos duplicados
  - Contém apenas os métodos definidos na interface Collection

## Abstract Set

```
public abstract class AbstractSet<E> extends AbstractCollection<E>
    implements Set<E> {

    protected AbstractSet();

    public boolean equals(Object o) {
        if (!(o instanceof Set)) return false;
        return ((Set)o).size()==size() && containsAll((Set)o);
    }

    public int hashCode() {
        int h = 0;
        for( E el : this )
            if ( el != null ) h += el.hashCode();
        return h;
    }
}
```

## HashSet

- Usa uma tabela de dispersão (Hash Map) para armazenar os elementos
- A inserção de um novo elemento não será efetuada se a função equals do elemento a ser inserido com algum elemento do Set retornar true. É fundamental implementar a função equals em todas as classes que possam ser usadas como elementos de tabelas de dispersão (HashSet, HashMap,...)
- Desempenho constante

```
public static void main(String args[]) {

    // vector para simular a entrada de dados no Set
    String[] str = {"Rui", "Manuel", "Rui", "Jose",
                  "Pires", "Eduardo", "Santos"};

    Set<String> group = new HashSet<>();
    for (String i: str ) {
        if (!group.add(i))
            System.out.println("Nome duplicado: " + i);
    }
    System.out.println(group.size() + " nomes distintos");

    for (String s: group)
        System.out.println( s );
}
```

Nome duplicado: Rui  
6 nomes distintos

Manuel  
Rui  
Jose  
Eduardo  
Santos  
Pires

Ordem?

**Conclusão:** sem noção de posição (sem ordem)

## TreeSet

- Permite a ordenação dos elementos pela sua “ordem natural”
- Implementação baseada numa estrutura em árvore balanceada
- Desempenho  $\log(n)$ , para add, remove e contains

### Exemplo 1:

```
import java.util.TreeSet;

public class Test {
    public static void main(String args[]) {
        TreeSet<String> ts = new TreeSet<>();
        ts.add("viagem");
        ts.add("calendário");
        ts.add("prova");
        ts.add("zircórnio");
        ts.add("ilha do sal");
        ts.add("avião");
        for (String element : ts)
            System.out.println(element + " ");
    }
}
```

avião  
calendário  
ilha do sal  
prova  
viagem  
zircórnio

### Exemplo 2:

```
public class TestTreeSet {

    public static void main(String[] args) {
        Collection<Quadrado> c = new TreeSet<>();
        c.add(new Quadrado(3, 4, 5.6));
        c.add(new Quadrado(1, 5, 4));
        c.add(new Quadrado(0, 0, 6));
        c.add(new Quadrado(4, 6, 7.4));
        System.out.println(c);

        for (Quadrado q: c)
            System.out.println(q);
    }
}
```

[Quadrado de Centro (1.0,5.0) e de lado 4.0, Quadrado de Centro (3.0,4.0) e de lado 5.6, Quadrado de Centro (0.0,0.0) e de lado 6.0, Quadrado de Centro (4.0,6.0) e de lado 7.4]  
Quadrado de Centro (1.0,5.0) e de lado 4.0  
Quadrado de Centro (3.0,4.0) e de lado 5.6  
Quadrado de Centro (0.0,0.0) e de lado 6.0  
Quadrado de Centro (4.0,6.0) e de lado 7.4

Ordem

- Mapas (Map)

- Estruturas associativas onde os objetos são representados por um par chave-valor
- Não descende de Collections ( Interface Map<K,V>)
- Um mapa é um conjunto que associa uma chave (K) a um valor (V)
- Não contém chaves duplicadas
- Também é denominado como dicionário ou memória associativa
- Métodos disponíveis:
  - . adicionar: put(K key, V value)
  - . remover: remove(Object key)
  - . obter um objeto: get(Object key)

Exemplo:

```
public interface Map<K,V> {  
    // Basic operations  
    V put(K key, V value);  
    V get(Object key);  
    V remove(Object key);  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
    // Bulk operations  
    void putAll(Map<? extends K, ? extends V> m);  
    void clear();  
    // Collection Views  
    public Set<K> keySet();  
    public Collection<V> values();  
    public Set<Map.Entry<K,V>> entrySet();  
    // Interface for entrySet elements  
    public interface Entry {  
        K getKey();  
        V getValue();  
        V setValue(V value);  
    }  
}
```



**Vistas**

## Vistas

- Mapas não são Collections
- No entanto, podemos obter vistas dos mapas
- As vistas são do tipo Collections
- Há 3 vistas disponíveis:
  - . conjunto (set) de chaves
  - . coleção de valores
  - . conjunto (set) de entradas do tipo par chave/valor

## HashMap

No [ArrayList](#) capítulo, você aprendeu que Arrays armazenam itens como uma coleção ordenada e você precisa acessá-los com um número de índice ( `int` tipo). Porém `HashMap`, armazene itens em pares " **chave** / **valor** " e você poderá acessá-los por um índice de outro tipo (por exemplo, a `String`).

Um objeto é usado como chave (índice) para outro objeto (valor). Pode armazenar diferentes tipos: `String` chaves e `Integer` valores, ou do mesmo tipo, como: `String` chaves e `String` valores:

### Exemplo 1:

```
// Import the HashMap class
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        // Create a HashMap object called capitalCities
        HashMap<String, String> capitalCities = new HashMap<String, String>();

        // Add keys and values (Country, City)
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");
        System.out.println(capitalCities);
    }
}
```

- Métodos disponíveis:

- . adicionar: put()
- . remover: remove()
- . acessar um item: get()
- . tamanho: size()
- . loop:

```
// Print keys
for (String i : capitalCities.keySet()) {
    System.out.println(i);
}
```

```
// Print values
for (String i : capitalCities.values()) {
    System.out.println(i);
}
```

```
// Print keys and values
for (String i : capitalCities.keySet()) {
    System.out.println("key: " + i + " value: " + capitalCities.get(i));
}
```

## Exemplo 2:

```
public static void main(String[] args) {
    Map<String, Double> mapa = new HashMap<>();
    mapa.put("Rui", 32.4);
    mapa.put("Manuel", 3.2);
    mapa.put("Rita", 5.6);

    System.out.println("O Mapa contém " + mapa.size() + " elementos");
    System.out.println("O Rui está no Mapa? " + mapa.containsKey("Rui"));

    System.out.println("A Rita tem " + mapa.get("Rita") + "€");
    mapa.put("Rita", mapa.get("Rita") + 3.6);
    System.out.println("A Rita tem " + mapa.get("Rita") + "€");

    Set<Entry<String, Double>> set = mapa.entrySet();
    for (Entry<String, Double> ele: set)
        System.out.println("O " + ele.getKey() + " ganha "
                           + ele.getValue() + "€");
}
```

O Mapa contém 3 elementos  
O Rui está no Mapa? true  
A Rita tem 5.6€  
A Rita tem 9.2€  
O Manuel ganha 3.2€  
O Rui ganha 32.4€  
O Rita ganha 9.2€

Vista



## TreeMap

- Mesmas características das descritas para a TreeSet mas adaptadas a pares key/value
- Oferece a possibilidade de ordenar objetos:
  - . utilizando a “Ordem Natural” (compareTo) ou um objeto do tipo Comparator
  - . utilização semelhante aos exemplos de HashSet

## ● Iterar sobre Coleções

- Um Iterator é um objeto que pode ser usado para percorrer coleções, como ArrayList e HashSet. É chamado de "iterador" porque "iterar" é o termo técnico para loop.
- Para usar um Iterator, você deve importá-lo do java.util.pacote.

### Exemplo 1:

```
// Import the ArrayList class and the Iterator class
import java.util.ArrayList;
import java.util.Iterator;

public class Main {
    public static void main(String[] args) {

        // Make a collection
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");

        // Get the iterator
        Iterator<String> it = cars.iterator();

        // Print the first item
        System.out.println(it.next());
    }
}
```

## Exemplo (Percorrer uma coleção):

```
while(it.hasNext()) {  
    System.out.println(it.next());  
}
```

## Exemplo (Removendo itens de uma coleção):

Use um iterador para remover números menores que 10 de uma coleção:

```
import java.util.ArrayList;  
import java.util.Iterator;  
  
public class Main {  
    public static void main(String[] args) {  
        ArrayList<Integer> numbers = new ArrayList<Integer>();  
        numbers.add(12);  
        numbers.add(8);  
        numbers.add(2);  
        numbers.add(23);  
        Iterator<Integer> it = numbers.iterator();  
        while(it.hasNext()) {  
            Integer i = it.next();  
            if(i < 10) {  
                it.remove();  
            }  
        }  
        System.out.println(numbers);  
    }  
}
```

## Último Exemplo:

```
public static void main(String args[]) {  
  
    // vector para simular a entrada de dados  
    String[] acessorios = {"Chinelos", "Toalha", "Protetor", "Prancha"};  
  
    List<String> saco = new ArrayList<>();  
    for (String obj: acessorios )  
        saco.add(obj);  
  
    // Iterador  
    Iterator<String> itr = saco.iterator();  
    while ( itr.hasNext() )  
        System.out.println( itr.next() );  
    // for  
    for (String s: saco)  
        System.out.println("\t"+s );  
}  
}
```

```
Chinelos  
Toalha  
Protetor  
Prancha  
  
    Chinelos  
    Toalha  
    Protetor  
    Prancha
```

- Ponto geral

Collections					
	Implementações				
Interfaces	Resizable array	Linked list	Hash table	Hash table + Linked list	Balanced Tree ( <u>sorted</u> )
List	ArrayList	LinkedList			
Queue	ArrayDeque	LinkedList			
Set			HashSet	LinkedHashSet	TreeSet
Map			HashMap	LinkedHashMap	TreeMap