Curso de Java

aula 07

Prof. Anderson Henrique

TECH
soluções em ti

ArrayList

A Java API fornece várias estruturas de dados predefinidas, chamadas **coleções**, usadas para armazenar grupos de objetos relacionados na memória.

Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem a necessidade de conhecer como os dados são armazenados.

Usamos arrays para armazenar sequências de objetos. Arrays não mudam automaticamente o tamanho em tempo de execução para acomodar elementos adicionais.

A classe de coleção ArrayList<**T**> fornece uma solução conveniente para esse problema – ela pode alterar dinamicamente seu tamanho para acomodar mais elementos. O **T** é um espaço reservado – ao declarar um novo ArrayList, substitua-o pelo tipo dos elementos que você deseja que o ArrayList armazene.

Ex.: ArrayList<String> list; ArrayList<Integer> integers;

A tabela a seguir mostra alguns métodos comuns da classes ArrayList:

Método	Descrição
add	Adiciona um elemento ao final do ArrayList
clear	Remove todos os elementos do ArrayList
contains	Retorna true se o ArrayList contém o elemento especificado, caso contrário, retorna false
get	Retorna o elemento no índice especificado
indexOf	Retorna o índice da primeira ocorrência do elemento especificado no ArrayList
remove	Sobrecarregado. Remove a primeira ocorrência do valor especificado ou o elemento no índice especificado
size	Retorna o número de elementos armazenados em ArrayList
trimtosize	Corta a capacidade do ArrayList para o número atual de elementos

Exemplo de um ArrayList

```
ArrayList<String> itens = new ArrayList<String>( );

itens.add(“vermelho”); //adiciona um item à lista
itens.add(0, “amarelo”); //adiciona no índice 0

System.out.println(“Mostrar os itens da lista com laço de repetição”);
for(int i = 0; i < itens.size( ); i++){
        System.out.println(itens.get[i]);
}
```

```
public static void mostrar(ArrayList<String> itens, String titulo){
        System.out.println(titulo);
        for(String item : itens){
                System.out.println(item);
                System.out.println( );
        }
mostrar(itens, “Mostrar os itens da lista utilizando for aprimorado”);
itens.add(“verde”);  //adiciona item ao fim da lista
itens.add(“amarelo”); //adiciona item ao fim da lista
mostrar(itens, “Lista com dois novos elementos”)
```

TECH
soluções em ti

```
itens.remove(“amarelo”); //remove a primeira ocorrência
mostrar(itens, “Remove a primeira ocorrência de amarelo”);

itens.remove(1); //remove item no índice 1
mostrar(itens, “Remove o segundo elemento da lista (verde)”);

System.out.printf(“\”vermelho\” %s está na lista”,
itens.contains(“vermelho”) ? “” : “não ”);

System.out.println(“Total de elementos: ”+itens.size( ));
```

Collection

As coleções Java são estruturas de dados, interfaces e métodos para manipular esses dados. Alguns exemplos de coleções: as cartas de baralho, suas músicas favoritas, os jogadores de um time de futebol.

A classe ArrayList é uma classe de coleção, e a interface Collection é a interface raiz na hierarquia das coleções: Set, List, Queue. Cada interface tem um objetivo específico.

A interface Set trabalha com registros sem duplicidade, ou seja, com registros únicos.

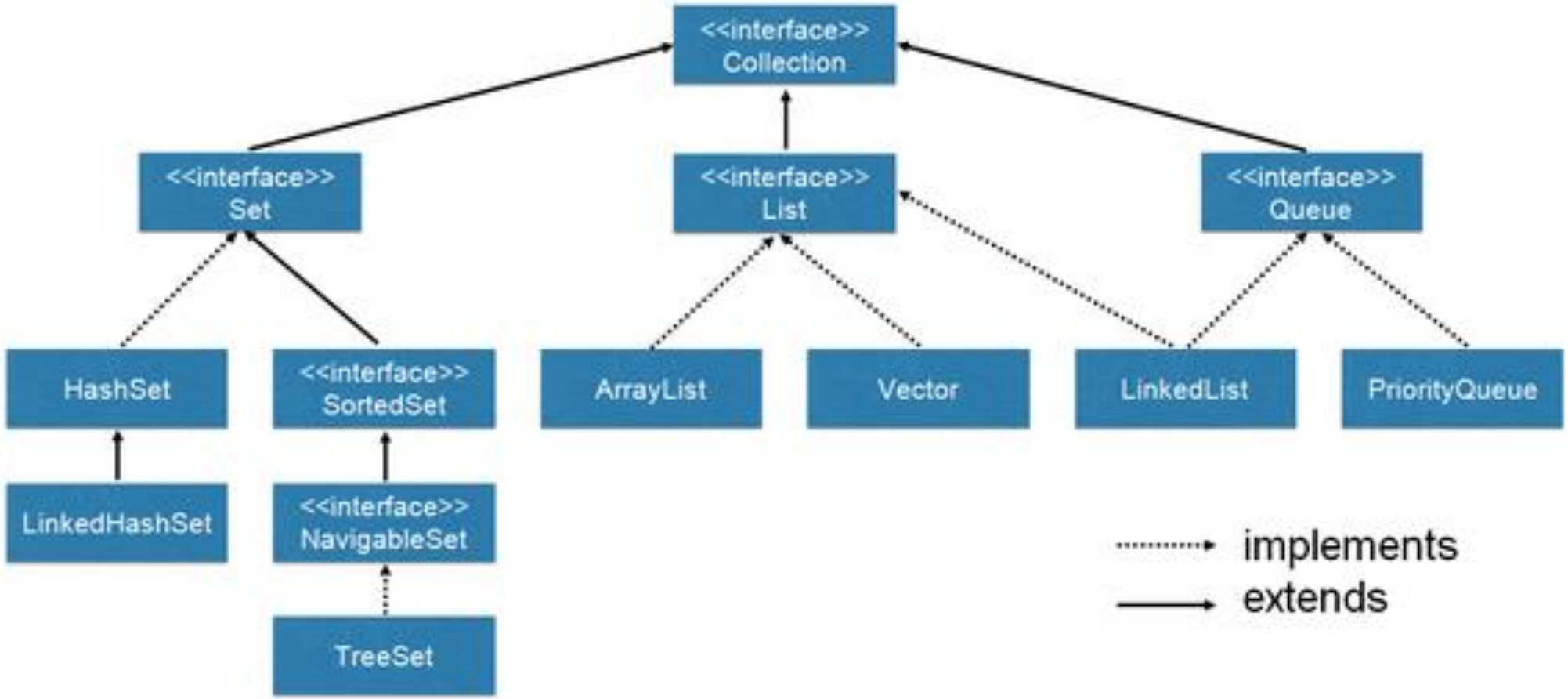
A interface List aceita registros duplicados e trabalha com esses registros na ordem em que eles são incluídos.

A interface Queue representa uma fila de espera, onde os registros são incluídos no final da fila e são removidos do início da fila.

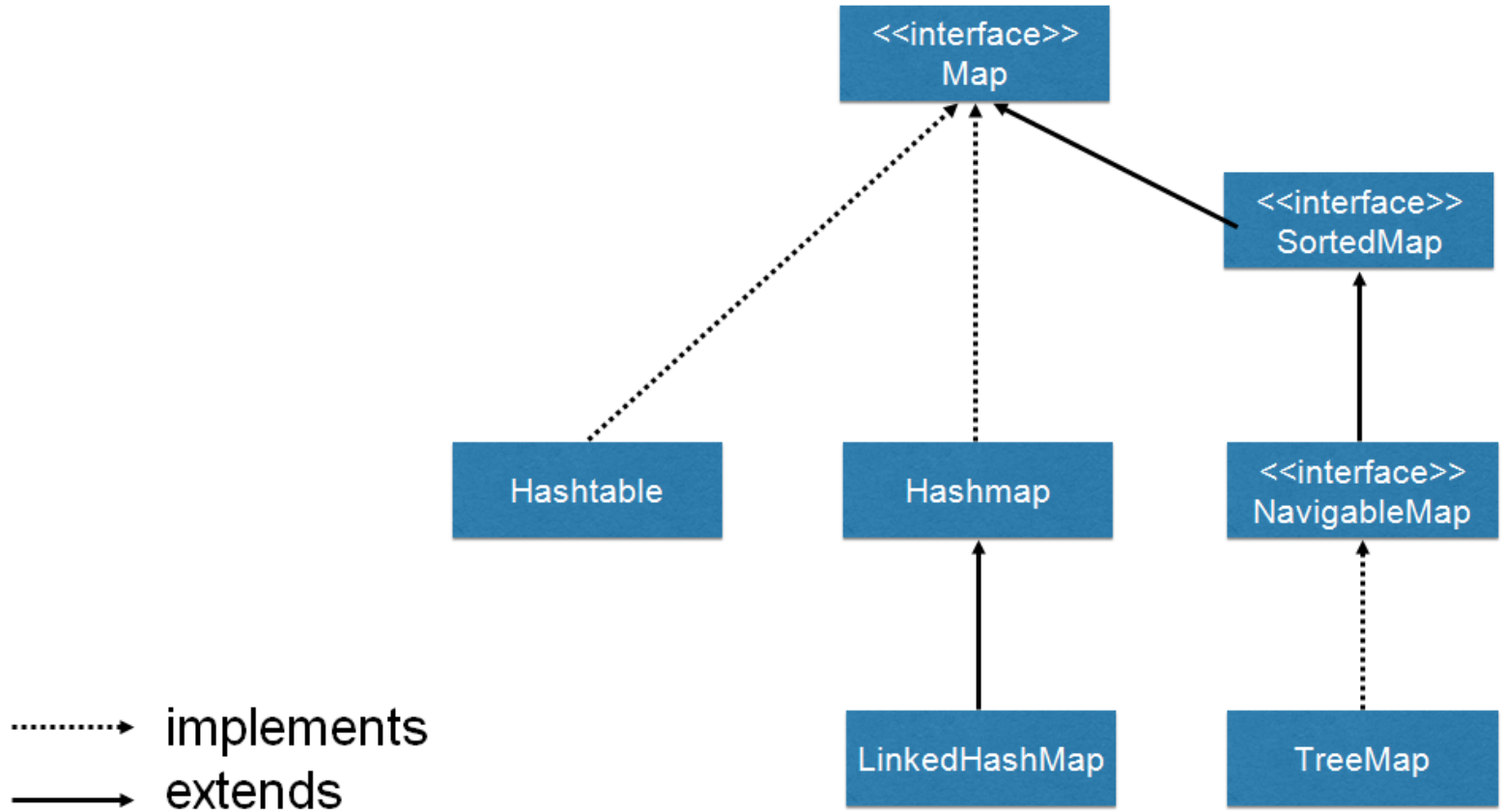
A interface Map trabalha com chave-valor, representa p.ex.: um dicionário, onde você tem a palavra e o significado da palavra.

Cada interface tem um conjunto de classes que a implementa, veja a figura a seguir:

Collection Interface



Map Interface



Além dessas coleções, temos as classes utilitárias: Arrays e Collections.

Ex.: Collection<String> c = new ArrayList<>( );

c.add(“A”); //adiciona elementos a coleção

c.add(“E”);

c.add(“I”);

c.toString( ); //apresenta coleção em forma de string

c.isEmpty( ); //verifica se coleção está vazia

c.contains(“E”); //verifica ocorrência de um determinado elemento

c.remove(“A”); //remove a primeira ocorrência do elemento

Métodos que trabalha com grupos

Ex.: Collection<String> c2 = Arrays.asList(“O”, “U”);

c.addAll(c2); //adiciona uma outra coleção

c.containsAll(c2); //verifica a ocorrência de todos os elementos de uma coleção

c.removeAll(c2); //remove todos os elementos de uma coleção

```
/* percorre os elementos */
For(String string : c){
        System.out.println(string);
}
```

```
/* converter coleção em um array */
Ex.:  String[ ] s = c.toArray(new String[c.size( )]);
/* limpa todos os elementos de uma coleção */
c.clear( );
```

Utilitário Collections

Ex.: List<String> frutas = new ArrayList<>( );

list.add(“Guaraná”);

list.add(“Uva”);

list.add(“Manga”);

list.add(“Coco”);

list.add(“Açaí”);

list.add(“Banana”);

A classe utilitária Collections serve para a manipulação de todo o tipo de coleção Java, possui métodos que só existem nessa classe.

Collections.sort(frutas); //ordenar alfabeticamente

Collections.reverse(frutas); //ordem reversa

Collections.shuffle(frutas); //ordem aleatória (embaralhar)

Collections.addAll(frutas, “Cupuaçu”, ”Laranja”, “Laranja”); //adiciona novos elementos

Collections.frequency(“Laranja”); //retorna quantidade de vezes que o elemento aparece na lista

List<String> list2 = Arrays.asList(“Acerola”, “Graviola”);

boolean d = Collections.disjoint(list, list2); //verifica se os elementos da list2 existem na list

Collections.sort(list); //ordenar alfabeticamente

int índice = Collections.binarySearch(list, “Guaraná”); //verifica em qual posição se encontra o elemento na lista

Collections.fill(list, “Açaí”); //adiciona em todas as posição o mesmo elemento

```
/* coleção que não pode ser mudada */
Collection<String> constante = Collections.unmodifiableCollection(list);
```

Collection List

Interface List aceita registros duplicados e trabalha com esses registros na ordem em que foram incluídos.

Ex.: List<String> lista = new ArrayList<>( );

lista.add(“Futebol”);

lista.add(“Basquete”);

lista.add(“Tênis”);

lista.add(“Natação”);

lista.add(“Hockey”);

lista.add(“Boxe”);

lista.add(“Futebol”);

```
System.out.println(lista);
/* percorre todos os elementos da lista */
for(int i = 0; i < lista.size( ); i++){
        String letra = lista.get(i); //recupera elemento na posição
        lista.set(i, letra.toUpperCase( )); //reatribuir elementos na mesma posição
}
System.out.println(lista);
lista.indexOf(“BOXE”); //retorna a posição do elemento dentro da lista
lista.sublist(2, 4); // retorna uma sublista
```

TECH
soluções em ti

```
/* remove uma sublista da lista principal */
lista.sublist(2, 4).clear( );
```

Collection Set

A interface Set trabalha com registros sem duplicidade, ou seja, com registros únicos.

Ex.: String[ ] cores = {“verde”, “amarelo”, “azul”, “branco”, “azul”, “amarelo”, “verde”};

```
List<String> list = Arrays.asList(cores);
System.out.println(list);

Set<String> set = new HashSet<>(list);
System.out.println(set);
```

Collection Queue

A interface Queue representa uma fila de espera, onde os registros são incluídos no final da fila e são removidos do início da fila.

Ex.: Queue<String> fila = new LinkedList<>( );

fila.add(“Anderson”); //adiciona elemento no final

fila.add(“Anna”);

fila.add(“Luciana”);

fila.offer(“Beatriz”); //adiciona elemento no final

Quero saber a próxima pessoa a ser atendida na fila, para isso eu tenho o método peek( )

```
fila.peek( );

fila.poll( ); //remove o elemento do início da fila


/* outros métodos disponíveis em linkedlist */

LinkedList<String> f = (LinkedList<String>) fila; //cast

f.addFirst(“Caio”); //adiciona no início da fila

f.addLast(“Juliana”); //adiciona no fim da fila

System.out.println(f);
```

```
f.peekFirst( ); //retorna a primeira pessoa da fila
f.peekLast( ); //retorna a última pessoa da fila

f.pollFirst( ); // remove a primeira pessoa da fila
f.pollLast( ); // remove a última pessoa da fila
```

Collection Map

A interface Map trabalha com chave-valor, representa p.ex.: um dicionário, onde você tem a palavra e o significado da palavra. As chaves devem ser únicas, mas os valores podem ser duplicados.

Ex.: Map<String, String> pais = new HashMap<>( );

pais.put(“BR”,”Brasil”); //adiciona elemento na coleção

pais.put(“RU”,”Rússia”);

pais.put(“IN”,”Índia”);

pais.put(“CN”,”China”);

System.out.println(pais);

TECH
soluções em ti

```
pais.containsKey(“BR”); //verifica se a coleção contém uma determinada chave
pais.containsKey(“US”); //verifica se a coleção contém uma determinada chave

pais.containsValue(“Brasil”); //verifica se a coleção contém um determinado valor
pais.get(“CN”); //recupera um conteúdo pela chave
pais.remove(“RU”) //remove um conteúdo pela chave

Set<String> Keys = pais.keySet( ); //recupera todas as chaves
for(String key : Keys){
        System.out.println(key + “:” + pais.get(key));
}
```

Prosseguiremos no próximo slide...

Professor: Anderson Henrique

Programador nas Linguagens Java e PHP