Java Collections

1

Coleções em Java

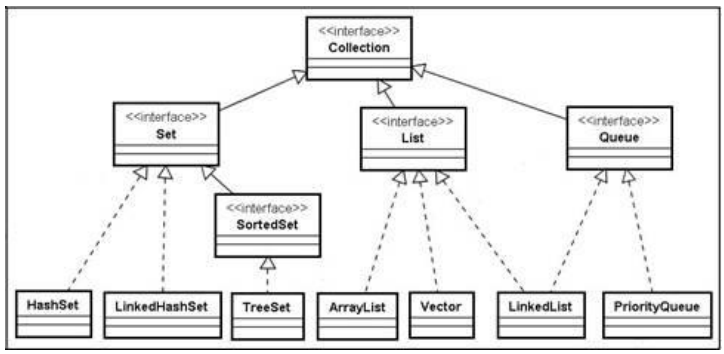
- Java fornece uma biblioteca de estruturas de dados utilizadas para manipular coleções de dados de forma robusta e otimizada;
- Ela atende aos principais propósitos de representação e manipulação de coleções de dados, tais como: listas, filas, conjuntos e listas associativas;
- Essa biblioteca está estruturada na forma de um framework;
- Esse framework é baseado em interfaces, implementações e classes auxiliares;
- Ele está contido no pacote `java.util`;
- A partir da versão 5.0 o *Java Collections Framework* implementa *Generics.*

2

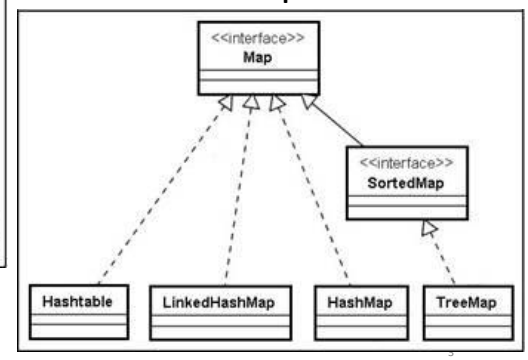
2

Java Collections Framework

Collection



Map



3

Collection - Classes de Implementação

ArrayList	Array armazenado de forma contígua e alocado dinamicamente
LinkedList	Array armazenado de forma encadeada o que facilita a inserção e remoção em qualquer posição
HashSet	Conjunto não ordenada que rejeita duplicação
TreeSet	Conjunto ordenado
EnumSet	Conjunto de tipos enumerados
LinkedHashSet	Conjunto que lembra a ordem em que os elementos foram inseridos
PriorityQueue	Coleção que permite eficiente remoção do elemento de maior prioridade

4

4

Map – Classes de Implementação

HashMap	Estrutura de dados que armazena associações entre chaves e valores
TreeMap	Um map com as chaves ordenadas
EnumMap	Um map com as chaves pertencentes a um tipo enumerado
LinkedHashMap	Um map que lembra a ordem que os elementos foram adicionados
WeakHashMap	Um map que pode ser coletado pelo "garbage collector" se não forem mais utilizados
IdentityHashMap	Um map que as chaves são comparadas por == e não por equals

Coleções e Generics

- Todo o *framework* de coleções foi adaptado na versão Java 5 para usar *generics*;
  - `Collection` → `Collection<T>`
  - `List` → `List<T>`
  - `Set` → `Set<T>`
  - `Map` → `Map<K, V>`
- Desta forma, usar um `ArrayList<String>` se torna tão simples quanto utilizar um array `String[]`;
- Entretanto, utilizar ArrayList é melhor que array porque o primeiro tem seu tamanho expandido automaticamente na inserção de um novo elemento.

Coleções e Generics

```
public void teste()
{
    List<String> lista = new ArrayList<String>();

    lista.add("Verde");
    lista.add("Amarelo");
    lista.add("Azul");
    lista.add("Branco");

    lista.add(new Integer(37)); //erro de compilação

    String s = lista.get(3);    //não precisa typecast
}
```


Collection

- Representa uma coleção de objetos.
- Seus principais métodos são:
  - `int size();`
    - Número de elementos da coleção
  - `boolean contains(Object elemento);`
    - Verifica se o parâmetro pertence à coleção
  - `boolean add(Object elemento);`
    - Adiciona o parâmetro à coleção
  - `boolean remove(Object elemento);`
    - Remove o parâmetro da coleção
  - `void clear();`
    - Remove todos os elementos da coleção
  - `Iterator<E> iterator();`
    - Cria um *Iterator* para iterar pelos elementos da coleção.

List

- List é uma coleção indexada de objetos.
- Possui os seguintes métodos além dos herdados de *Collection*:
  - `E get(int indice);`
    - Acessa o i-ésimo elemento da lista
  - `void set(int indice, E elemento);`
    - Altera o i-ésimo elemento da lista
  - `void add(int indice, E elemento);`
    - Adiciona um elemento na posição i da lista. Se havia elementos após este índice, eles serão movidos. Se os índice for maior que o tamanho da lista será lançada uma exceção.
  - `E remove(int index);`
    - Remove o i-ésimo elemento da lista
  - `int indexOf(Object o);`
    - Obtém o índice do elemento passado como parâmetro

List – ArrayList x LinkedList

- *List* possui duas implementações principais:
  - *ArrayList*
    - Elementos são armazenados de forma contígua, como em um array.
    - Acesso indexado rápido.
    - Inserções e remoções no meio da lista são lentos.
  - *LinkedList*
    - Elementos são armazenados na forma de uma lista encadeada.
    - Acesso indexado é lento, pois precisa percorrer toda a lista.
    - Inserções e remoções no meio da lista são rápidos.

ArrayList – Exemplo

```
public void teste()
{
    List<Integer> lista = new ArrayList<Integer> ();

    lista.add(2);
    lista.add(7);
    lista.add(3);

    for (int i=0; i<lista.size(); i++)
        System.out.println( lista.get(i) );

    int idx = lista.indexOf(7);

    lista.set(idx, 5);
}
```


Set / HashSet

- *Set* possui os mesmos métodos que *Collection*.
- A diferença é que classes que implementam *Set* não possuem objetos repetidos.
- É usado o método *equals* para verificar objetos repetidos.
- A principal implementação de Set é a classe *HashSet*.

Set / HashSet

```
public void teste() {
    Set<Integer> s = new HashSet<Integer>();

    s.add(2);
    s.add(7);
    s.add(3);
    s.add(3); // este não é adicionado.

    if (s.contains(7) {
        s.remove(7);
    }
    else {
        s.add(1);
        s.add(4);
    }
}
```


Iterator

- O *Iterators* é utilizado para percorrer os elementos de uma coleção;
- O *iterator* inicia apontando para o primeiro elemento da coleção;
- Ele possui os seguintes métodos:
  - `boolean hasNext();`
    - indica se existe mais algum elemento que ainda não foi apontado.
  - `E next();`
    - Faz o *Iterator* apontar para o próximo elemento, e retorna este. Lança *NoSuchElementException*, caso não haja mais elementos.
  - `void remove();`
    - Remove o elemento atualmente apontado da coleção.

Comando *for-each* (para-cada)

- Permite iterar sobre *arrays* e coleções de forma simplificada.

```
public void teste2()
{
    int[] array = new int[4];


    array[0] = 10;
    array[1] = 20;
    array[2] = 30;
    array[3] = 40;

    for (int i : array) {
        System.out.println(i);
    }
}
```

```
public void teste1()
{
    List<String> lista =
       new ArrayList <String>();

    lista.add("Verde");
    lista.add("Amarelo");
    lista.add("Azul");
    lista.add("Branco");

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

15

15

Set – Exemplo usando *while*

```
public void teste() {
    Set<Integer> s = new HashSet<Integer>();

    s.add(2);
    s.add(7);
    s.add(3);

    Iterator it = s.iterator();
    while (it.hasNext()) {
        Integer i = it.next();

        if (i.intValue() > 2)
            System.out.println(i);
    }
}
```

16

16

Set – Exemplo usando *for*

```
public void teste() {
    Set<Integer> s = new HashSet<Integer>();

    s.add(2);
    s.add(7);
    s.add(3);

    for (Iterator it = s.iterator(); it.hasNext(); ) {
        Integer i = it.next();

        if (i.intValue() > 2)
            System.out.println(i);
    }
}
```

17

17

Set – Exemplo usando *for-each*

```
public void teste() {
    Set<Integer> s = new HashSet<Integer>();

    s.add(2);
    s.add(7);
    s.add(3);

    for (Integer i : s) {
        if (i.intValue() > 2)
            System.out.println(i);
    }
}
```

18

18

Map

- *Map* é uma coleção associativa;
- Valores inseridos devem ser associados a uma chave única;
- Esta chave é usada para obter novamente o valor;
- A principal implementação de *Map* é *HashMap;*
- Principais métodos:
  - `V put(K key, V value);`
    - Adiciona o objeto *value*, associado com *key*
  - `V get(K key);`
    - Acessa o objeto associado com *key*
  - `boolean remove(Object key);`
    - Remove o objeto associado com *key*
  - `int size();`
    - Número de elementos do *Map*

Map – Sub-coleções

- `Set<K> keySet();`
  - Acessa o conjunto das chaves do *Map*
- `Collection<V> values();`
  - Acessa a coleção de valores do *Map*
- `Set<Map.Entry<K, V>> entrySet();`
  - Acessa o conjunto de entradas *Map*

Map – Exemplo usando *while*

```
class Pessoa { ... }

public void teste() {
     Map<String, Pessoa> m = new HashMap<String, Pessoa>();

     m.put("Fabiano", new Pessoa("Fabiano", "Baldo", 30));
     m.put("Leandro", new Pessoa("Leandro", "Loss", 28));
     m.put("Rui", new Pessoa("Rui", "Silva", 35));

     Iterator it = m.values().iterator();
     while (it.hasNext()) {
         Pessoa p = (Pessoa) it.next();

         System.out.println(p.getNome());
     }
}
```

21

21

Map – Exemplo usando *for*

```
public class Pessoa { ... }

public void teste() {
     Map<String, Pessoa> m = new HashMap<String, Pessoa>();

     m.put("Fabiano", new Pessoa("Fabiano", "Baldo", 30));
     m.put("Leandro", new Pessoa("Leandro", "Loss", 28));
     m.put("Rui", new Pessoa("Rui", "Silva", 35));

     for (Iterator it = m.values().iterator(); it.hasNext();) {
         Pessoa p = (Pessoa) it.next();

         System.out.println(p.getNome()+" "+p.getSobrenome());
     }
}
```

22

22

Map – Exemplo usando *for-each*

```
class Pessoa { ... }

public void teste() {
     Map<String, Pessoa> m = new HashMap<String, Pessoa>();

     m.put("Fabiano", new Pessoa("Fabiano", "Baldo", 30));
     m.put("Leandro", new Pessoa("Leandro", "Loss", 28));
     m.put("Rui", new Pessoa("Rui", "Silva", 35));

     for (Pessoa p : m.values()) {

          System.out.println(p.getNome()+" "+p.getSobrenome());
     }
}
```


Collections – Métodos Utilitários

- O *framework* de coleções possui a classe *Collections* com algoritmos genéricos e métodos utilitários.
  - `void sort(List list);`
  - `void reverse(List list);`
  - `void shuffle(List list);`
  - `Object min(Collection coll);`
  - `Object max(Collection coll).`

Collections – Métodos Utilitários

```
public void teste() {
    List<String> lista = new ArrayList<String>();

    lista.add("Verde");
    lista.add("Amarelo");
    lista.add("Azul");
    lista.add("Branco");

    System.out.println(lista);

    Collections.sort(lista);
    System.out.println(lista);

    Collections.reverse(lista);
    System.out.println(lista);

    Collections.shuffle(lista);
    System.out.println(lista);

    String s = Collections.min(lista);
    System.out.println("Mínimo = " + s);
}
```

25

25

Collections – Adaptadores

- *Collections* possui ainda métodos para gerar adaptadores não-modificáveis de outras coleções;
- Qualquer operação que modificaria a coleção retornada por um destes métodos gera uma *UnsupportedOperationException.*
- Operações não modificantes são delegadas para a coleção original;
- Não é feita nenhuma cópia de objetos, ou seja, não há problema de desempenho;
  - `Collection unmodifiableCollection(Collection c);`
  - `Set unmodifiableSet(Set s);`
  - `List unmodifiableList(List list);`
  - `Map unmodifiableMap(Map m).`

26

26

Collections – Adaptadores

```
public void teste() {
    List<String> lista = new ArrayList<String>();

    lista.add("Verde");
    lista.add("Amarelo");
    lista.add("Azul");
    lista.add("Branco");

    List lista2 = Collections.unmodifiableList(lista);

    String s = (String) lista2.get(3); //ok

    lista2.add("Vermelho");   //exceção
}
```