Collections

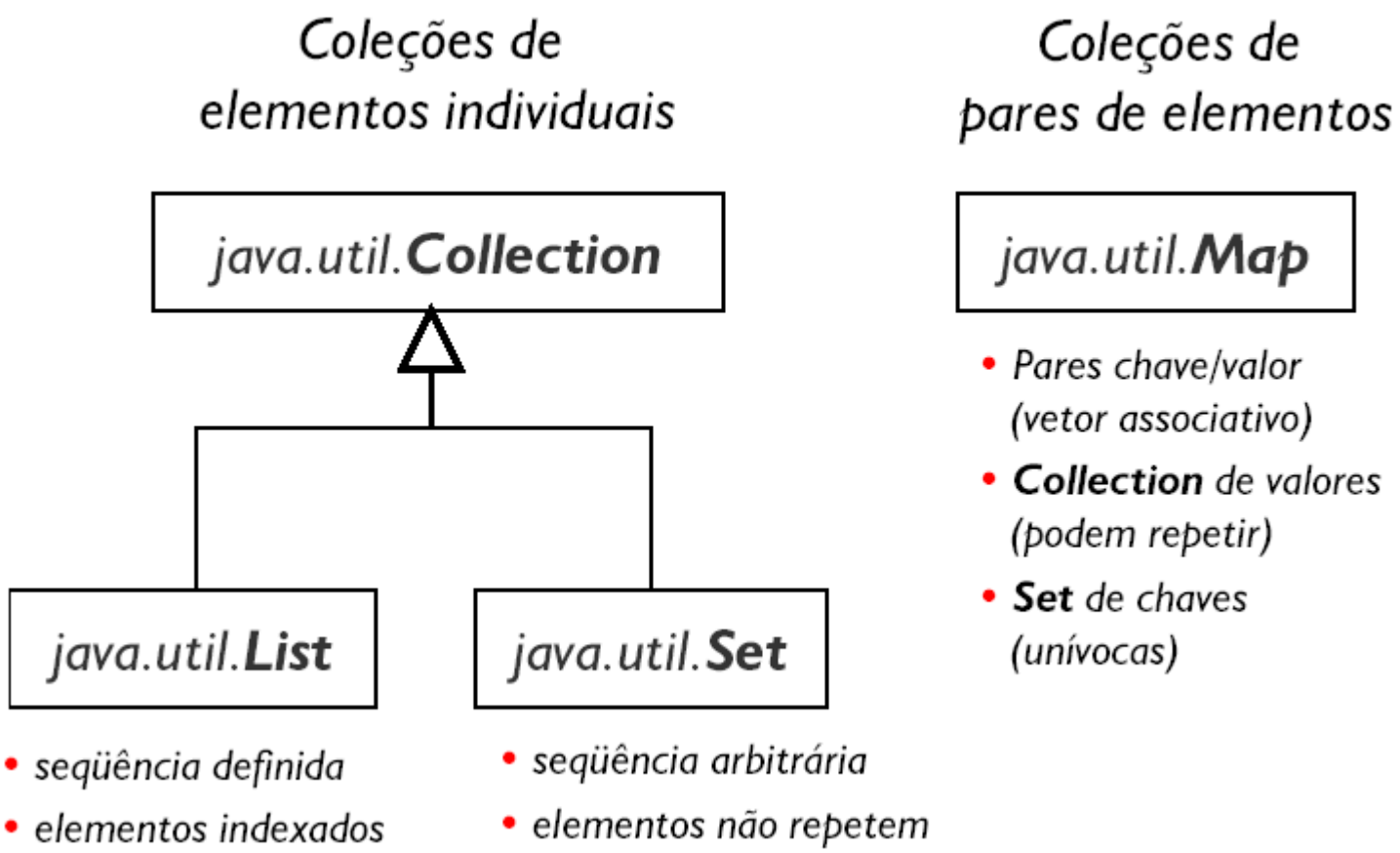
Prof. Ricardo P. Mesquita

Coleções e Mapas

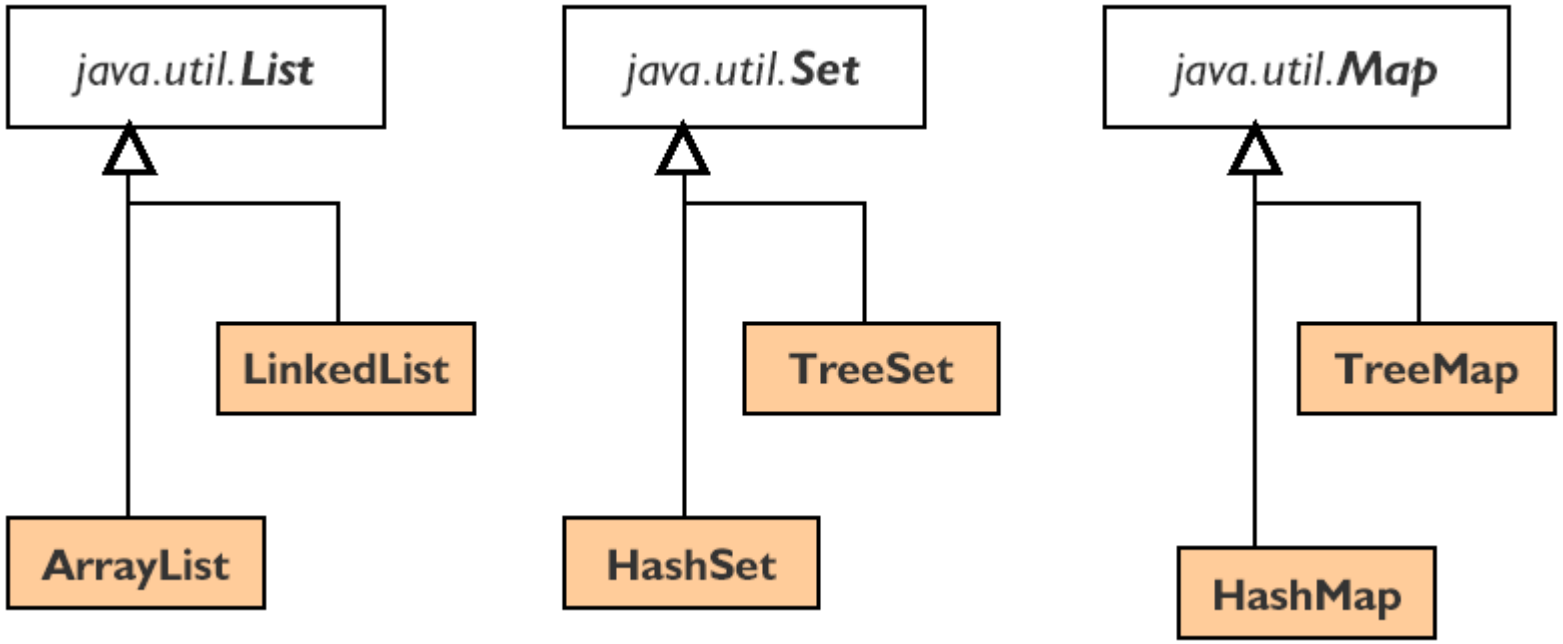
- Mecanismo que permite o armazenamento de vários objetos.
- É uma solução flexível e simples.
- O pacote **java.util** oferece duas categorias de estruturas de dados ‘genéricas’
  - **Coleções**: implementam ‘coleções’ de objetos sob a forma de *listas* e *conjuntos*.
  - **Mapas**: mantém pares de objetos na forma (*chave*, *valor*).

Obs: Não aceitam tipos primitivos (int, short, long etc.)

Coleções e Mapas



Coleções e Mapas



A Interface Collection

- Define os métodos disponíveis para operar com as ‘coleções’ de objetos:
  - size()
  - isEmpty()
  - clear()
  - add(Object)
  - contains(Object)
  - remove(Object)
  - containsAll(Collection)
  - removeAll(Collection)
  - retainAll(Collection)
  - toArray()

A Interface List

- Define os métodos para operações com listas:
  - get(int)
  - set(Object,int)
  - add(Object,int)
  - remove(int)
  - indexOf(Object)

ArrayList e LinkedList

- Implementam a interface ***List***:
  - ***ArrayList***: implementa uma lista de objetos num vetor cujo tamanho pode variar dinamicamente (!).
    - ***ArrayList*** é mais adequada em situações onde o acesso aleatório aos elementos é mais frequente. A implementação do vetor de 'tamanho variável' é cara.
  - ***LinkedList***: implementa uma lista de objetos sob a forma de uma lista encadeada.
    - ***LinkedList*** é mais adequada em casos onde o acesso aleatório não é frequente e o tamanho da lista pode variar muito.

A Interface Set

- Define os métodos para operações com conjuntos de objetos (um conjunto não pode conter elementos repetidos):
  - add(Object)
  - remove(Object)
  - contains(Object)
  - addAll(Set)
  - retainsAll(Set)
  - removeAll(Set)

A Interface Set

- Implementações da interface Set
  - HashSet: baseada em *tabela de hashing*.
  - TreeSet: baseada em árvore binária de busca balanceada.

Mapas

- O Map tem uma combinação de chaves/valores.
- Uma chave está associada a um valor.
- As chaves, claro, são únicas.
- Os valores podem ser duplicados.

Mapas

- A interface Map define os seguintes métodos:
  - put(Object key, Object value)
  - get(Object key)
  - putAll(Map)
  - remove(Object key)
  - contains(Object key)
  - size()
  - isEmpty()
  - clear()

Mapas

- Classes que implementam a interface Map
  - HashMap – baseada em tabela *hash*.
  - TreeMap – baseada em árvore binária de busca balanceada.
  - LinkedHashMap – combinação de tabela *hash* e lista ligada.

A Interface Iterator

- A interface Iterator define métodos para percorrer Collections:
  - iterator(): As classes que implementam a interface Collection oferecem este método que deve retornar um objeto Iterator para a coleção.
  - hasNext(): devolve o valor true se ainda existir objeto a ser percorrido na coleção.
  - next(): devolve o próximo objeto da coleção.
  - remove(): remove um elemento

Exemplo: Iterator

```
...
String[] names = {
                    "um","dois","tres",
                    "quatro","cinco","seis"
                  };
...
ArrayList list = new ArrayList();
...
for(int i = 0; i < names.length; i++) list.add(names[i]);
...
Iterator it = list.iterator();
while(it.hasNext()){
   System.out.println("==> "+(String)it.next());
}
```

Exemplo: HashMap

```
...
HashMap map = new HashMap();
...
for(int i = 0; i < names.length; i++)
  map.put(names[i],new Xbluft(names,i,etc));
...
Iterator it = map.keySet().iterator();
while(it.hasNext()){
   Xbluft xb = (Xbluft)map.get(it.next());
   System.out.println("==>"+xb.toString());
}
```

Exemplos:

- Vamos **explorar** os seguintes programas-exemplo:
  1. Interface Collection demonstrada por meio de um objeto ArrayList: **CollectionTest.java**
  2. Lists, LinkedLists e ListIterators: **ListTest.java**
  3. Visualizando arrays como Lists e convertendo Lists em arrays: **UsingToArray.java**
  4. O método Collections sort: **Sort1.java**
  5. Método sort decrescente: **Sort2.java**
  6. Classe Comparator personalizada que compara dois objetos Time2: **Pasta Exemplo6**
  7. Embaralhamento e distribuição de cartas com método shuffle de Collections: **DeckOfCards.java**
  8. Métodos Collections reverse, fill, copy, max e min: **Algorithms1.java**
  9. Método Collections binarySearch: **BinarySearchTest.java**
  10. Métodos Collections addAll, frequency e disjoint: **Algorithms2.java**

Exemplos

- Vamos **explorar** os seguintes programas-exemplo:
  11. Pilha (classe Stack): StackTest.java
  12. Filas de prioridade (classe PriorityQueue): PriorityQueueTest.java
  13. HashSet utilizado para remover valores duplicados do array de strings: SetTest.java
  14. Usando SortedSets e TreeSets: SortedSetTest.java
  15. O programa conta o número de ocorrências de cada palavra em uma String: WordTypeCount.java
  16. Classe Properties: PropertiesTest.java

Dúvidas?