Coleções

prof. Angelo C. Loula

Coleções

- Collection/Container
- Um objeto que agrupa vários elementos em uma única unidade
- Collection Framework
  - Interfaces: Tipos Abstratos de Dados
  - Implementações: Implementações concretas das interfaces
  - Algoritmos: computações em coleções, ex busca e ordenação

Coleções

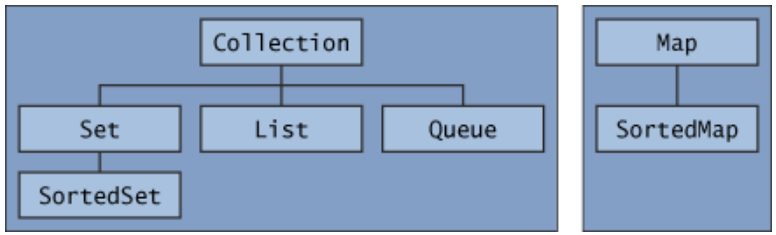
- Vantagens
  - Facilita a programação
  - Melhor compreensão
  - Uniformiza acesso
  - Interoperabilidade
  - Reutilização

Coleções

- Tipo Abstrato de Dados
  - Modelo para estruturas de dados com comportamento semelhante
  - Especificação independe de uma implementação em algoritmo
  - Ex: Pilha
    - Operações: push e pop
    - Implementações: array ou lista ligada

Coleções

- Interfaces



Coleções

- Interfaces
  - Collection: grupo genérico de elementos
  - Set: conjunto, sem ordem e sem duplicação
  - List: existe ordem, sequência, pode duplicar elementos
  - Queue: métodos especiais de inserção, remoção e inspeção, pode ser FIFO
  - Map: pares de chave/valor, sem chaves duplicadas e cada chave tem 0 ou 1 valor associado


```
public interface Collection<E> extends Iterable<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```


Coleções

- Collection
  - for-each

```
for (Object o : collection)
    System.out.println(o);
```

  - Interator
    - objeto para varrer uma coleção

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

  - Convertendo para Array

```
Object[] a = c.toArray();
String[] a = c.toArray(new String[0]);
```

Coleções

- Set
  - Implementação:
    - HashSet
      - Tabela hash
      - Utiliza método hashCode() dos elementos
        - » sobrescreva hashCode()!
    - TreeSet
      - árvore rubro-negra, ordena pelo valor do elementos
    - LinkedHashSet
      - acrescenta ligações duplas entre elementos da tabela hash (mantém ordem de inserção)

Coleções

- HashCode
  - hashCode padrão é tipicamente o endereço interno do objeto convertido para inteiro, mas não é garantido
  - hashCode() de String é dado pelo hash de Horner com núm. primo 31

$$h = s[0] * 31^{n-1} + s[1] * 31^{n-2} + \cdots + s[n-1]$$

s[i] i-ésimo caracter e n tamanho da string

  - hashCode deve ser compatível com equals
    - o1.equals(o2)==true → o1.hashcode()==o2.hashcode()

Coleções

- List

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index,
        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);
}
```


Coleções

- List
  - Implementações
    - ArrayList
      - Array redimensionável
      - Configurações: tamanho inicial e tamanho mínimo
    - LinkedList
      - Lista duplamente ligada
      - Pilha, Fila, Fila dupla
      - Métodos próprios para manipulação do primeiro e último elemento
    - Vector (alterado para ser compatível)
      - Configurações: tamanho inicial, tamanho mínimo, incremento de capacidade

Coleções

- List
  - Iterador

```
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
}
```


Coleções

- Qeue

```
public interface Queue<E> extends Collection<E> {
    E element();
    boolean offer(E e);
    E peek();
    E poll();
    E remove();
}
```

  - Implementações:
    - LinkedList e PriorityQeue

Coleções

- Map
  - Modela a abstração de função matemática
  - Implementações
    - HashMap, TreeMap, LinkedHashMap
      - Funcionamento similar às implementações de Set
    - Hashtable (modificado para ser compatível)


```
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);
    }
}
```

Coleções

- Map
  - Visão dos elementos

```
keySet, values, entrySet
for (KeyType key : m.keySet())
    System.out.println(key);

for (Iterator<Type> it =
 m.keySet().iterator(); it.hasNext(); )
    System.out.println(it.next());
```


Coleções

- Ordenação: Collections.sort(mylist)
- Tipos básicos já tem ordem natural
- Outros tipos de dados:
  - Comparable

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

  - Comparator

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

  - compare e compareTo retornam valor==0, >0 ou <0
  - Cuidado: e1.compareTo(e2) == 0 deve ser o mesmo que e1.equals(e2)


```
public class Name implements Comparable<Name> {
    private final String firstName, lastName;
    ...
    public int compareTo(Name n) {
        int lastCmp =
            lastName.compareTo(n.lastName);
        return (lastCmp != 0 ? lastCmp :
                firstName.compareTo(n.firstName));
         //falta tratar atributos null!
    }
    public boolean equals(Object o) {
        if (!(o instanceof Name))
            return false;
        Name n = (Name)o;
        return n.firstName.equals(firstName) &&
               n.lastName.equals(lastName);
    }
    public int hashCode() {
        return 31*firstName.hashCode() + lastName.hashCode();
    }
    public String toString() {
       return firstName + " " + lastName;
    }
```


Coleções

- SortedSet
  - Set ordenado pelo valor dos elementos

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Coleções

- SortedMap
  - Map ordenado pelo valor das chaves

```
public interface SortedMap<K, V> extends Map<K, V>{
    Comparator<? super K> comparator();
    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();
}
```


Coleções

- Implementações de Collections

General-purpose Implementations					
Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

- Tente declarar sempre que possível com a interface e não com a implementação

Coleções

- Algoritmos
  - Métodos do pacote Collections
  - A maioria atua sobre List
  - Alguns podem atuar em qualquer coleção

Coleções

- Algoritmos: Métodos do pacote Collections (Lista)
  - sort: merge sort
  - shuffle: permuta elementos aleatoriamento
  - reverse: reverte ordem dos elementos
  - rotate: rotaciona elementos em dada distância
  - Swap: troca elementos de posição
  - replaceAll: troca todas ocorrência de um elemento por outro
  - fill: sobreescreve todos elementos com o elemento dado
  - copy: copia de uma lista para outra
  - binarySearch: busca binária em lista ordenada
  - indexOfSubList e lastIndexOfSubList: retorna indice onde ocorre uma sublista em uma lista

Coleções

- Algoritmos
  - Métodos do pacote Collections (todos)
    - frequency: conta quantas vezes um elemento ocorre na coleção
    - disjoint: verifica de suas coleções não possuem elementos em comum
    - min: menor valor
    - max: maior valor