JAC444 - Introduction to Java for C++ Programmers

Lesson 12: Java Collections

Agenda

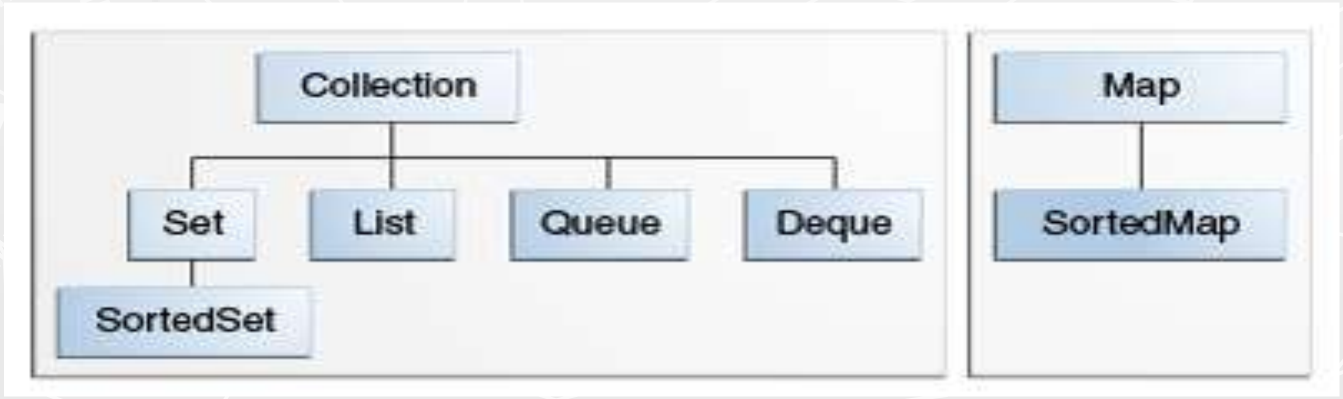
- Collections
  - Java Collections Framework.
  - The Collection Interface.
  - Set, List, Map Interfaces.

Introduction

- A collection – sometimes called a container - represents a group of objects, which are known as its elements.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.
- All collections frameworks contain the following:
    - Interfaces
    - Implementations
    - Algorithms

The Collection Interface

- Core collection interfaces
  - the Collection interface
    - the Set interface
    - the List interface
    - the Queue interface
  - the Map interface



The Collection Interface

- an abstraction of a group of objects (i.e. elements)
- used with specific implementation classes (e.g. HashSet, LinkedList, PriorityQueue)
- the AbstractCollection class
  - the size and iterator methods (not implemented)
- the use of an Iterator object to visit the elements in a Java collection
- an enhanced for loop (Java 1.5, 1.6)
  - for (Double element: arrayList) {…}
- package name: java.util

The Collection Interface

```
public interface Collection {
        // Group 1
        int size();
        boolean isEmpty();
        boolean contains(Object element);
        boolean add(Object element);    // Optional
        boolean remove(Object element); // Optional
        Iterator iterator();

        // Group 2
        boolean containsAll(Collection c);
        boolean addAll(Collection c);    // Optional
        boolean removeAll(Collection c); // Optional
        boolean retainAll(Collection c); // Optional
        void clear();                    // Optional

        // Group 3
        Object[] toArray();
        Object[] toArray(Object a[]);
    }
```

1 Basic Operations

2 Bulk Operations

3 Array Operations

Interface Iterator

- An object that implements the Iterator interface   generates a series of elements, one at a time.

```
public interface Iterator   {
          boolean hasNext();
          Object next();
          void remove(); // Optional
}
```

The List Interface

- A List is an ordered Collection (sometimes called a sequence)
- Lists may contain duplicate elements.
- implementation classes
  - ArrayList
  - LinkedList

    e.g. LinkedListDemo.java

    LinkedListDemo_v5.java, (using generics)
  - Vector

    Difference between *ArrayList and Vector* In *java*

The List Interface

```
public interface List extends Collection {
        // Positional Access
        Object get(int index);
        Object set(int index, Object element);              // Optional
        void add(int index, Object element);                // Optional
        Object remove(int index);                           // Optional
        abstract boolean addAll(int index, Collection c); // Optional

        // Search
        int indexOf(Object o);
        int lastIndexOf(Object o);

        // Iteration
        ListIterator listIterator();
        ListIterator listIterator(int index);

        // Range-view
        List subList(int from, int to);
    }
```

1 Access

2 Search

3 Iteration

4 Range

List Iterator

```
public interface ListIterator extends Iterator {
          boolean hasNext();
          Object next();

          boolean hasPrevious();
          Object previous();

          int nextIndex();
          int previousIndex();

          void remove();          // Optional
          void set(Object o);     // Optional
          void add(Object o);     // Optional
      }
```

e.g. LinkedListDemo.java, LinkedListDemo_v5.java

The Set Interface

- no duplicate elements are allowed in a Set.
- implementation classes
  - HashSet

    e.g. HashSetDemo.java, (using generics)

    HashSetDemo_v5.java
  - TreeSet (the elements are sorted)
  - LinkedHashSet (the elements are ordered by the way they are inserted)
- Difference between *TreeSet, LinkedHashSet* and *HashSet*

The Set Interface

HashSet

TreeSet

```
public interface Set {
          // Group 1
          int size();
          boolean isEmpty();
          boolean contains(Object element);
          boolean add(Object element);    // Optional
          boolean remove(Object element); // Optional
          Iterator iterator();

          // Group 2
          boolean containsAll(Collection c);
          boolean addAll(Collection c);    // Optional
          boolean removeAll(Collection c); // Optional
          boolean retainAll(Collection c); // Optional
          void clear();                    // Optional

          // Group 3
          Object[] toArray();
          Object[] toArray(Object a[]);
        }
```

1 Basic Operations

2 Bulk Operations

3 Array Operations

The Map Interface

- A Map is an object that maps keys to values/elements
  - distinct keys
- implementation classes
  - Hashtable (prior to JDK 1.2)
    - e.g. HashtableDemo2.java
  - HashMap (elements are not ordered)
  - TreeMap (keys are sorted)
  - LinkedHashMap (elements are ordered)

The Map Interface

```
public interface Map {
        // Basic Operations
        Object put(Object key, Object value);
        Object get(Object key);
        Object remove(Object key);
        boolean containsKey(Object key);
        boolean containsValue(Object value);
        int size();
        boolean isEmpty();

        // Bulk Operations
        void putAll(Map t);
        void clear();

        // Collection Views
        public Set keySet();
        public Collection values();
        public Set entrySet();

        // Interface for entrySet elements
        public interface Entry {
           Object getKey();
           Object getValue();
           Object setValue(Object value);
        }
    }
```

1 Basic

2 Bulk

3 View

4 Entry Interface

Sorting a Java collection

- Collections.sort( )

  e.g. SortDemo_v5.java

- Advanced feature:
  - Both TreeSet and TreeMap store elements in sorted order. However, what *sorted order* means for objects in TreeSet or TreeMap?
    - it is the comparator that defines sorted order.

The Comparator Interface

- The Comparator Interface used to compare two objects
- It defines two methods
  - int compare(Object obj1, Object obj2)
  - boolean equals(Object obj)

- a comparison class that implements the interface
  - e.g. compare the areas of two geometric objects
  - e.g. CompareToDemo.java
  -

The Queue Interface

- FIFO data structures
- by the order of insertion
  - the most recently inserted element
- by the order of priority
  - the element with the highest priority (the least value)

e.g. PriorityQueueDemo.java

The Queue Interface

- Each Queue method exists in two forms:
  1) one throws an exception if the operation fails.
  2) the other returns a special value if the operation fails (e.g. null or false).

Type of Operation	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

Resourceful Links

- Collections (The Java™ Tutorials)

Thank You!