Software Construction
Java Collections

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

ILOs

- Motivation for collections
- How they are implemented (brief)
- How to use collections in your code

Java Collections: What and Why

What are they: An object that groups multiple elements into a single unit

- to store and manipulate data
- to transmit data from one method to another

Why use them: By using collections/containers we can:

- reduce programming effort
- increase code quality
- rapid development of code

Java Collections Framework JCF

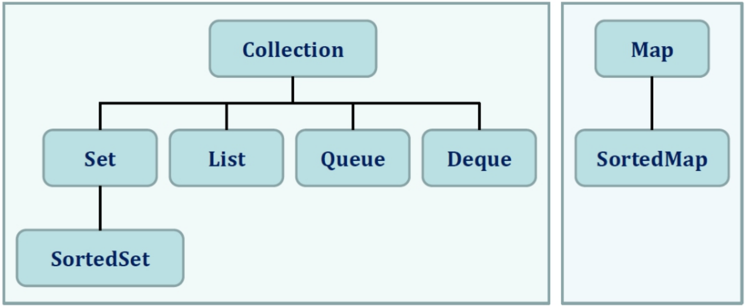
Gives an unified architecture for representing and manipulating collections. Which includes:

- interface: which specifies how the collection should be manipulated
- implementation: implementation of the said interface (in a suitable manner, which you do not have to worry about)
- algorithms: useful algorithms to perform on the objects in collection.

Basics

- You can find Java Collections in Package *java.util.*
- We will look at only a few interfaces
- See http://docs.oracle.com/javase/tutorial/collections/index.html

Collection Interface Hierarchy



- You will have different implementations of the same type
- Example: Sets can be *SortedSet*, *HashSet*, *TreeSet*
- All different implementations have the same interface (but will have different performances)

Types of Collections

Java supplies several types of Collections

- Set: No duplicate elements, order is not important
- SortedSet: like a set, but order is important
- List: may contain duplicate elements, order is important
- Queue: FIFO or a priority based removal

Also some "Collection-like" things

- Map: a dictionary for key-value pairs that can be access using keys
- SortedMap: like a map, but order is important

Interface: Collection

- A collection represents a group of objects known as elements
- Primary use: to pass around collections of objects
- One can create a different collection using an existing one

Java.util.Collection<E>

- *public int size()*: return number of elements in collection
- *public boolean isEmpty()*: return true if collection has no elements
- *public boolean add(E x)*: add the new element x to the collection. If the collection is changed return true
- *public boolean contains(Object x)*: returns true if and only if collection contains x (uses equals method)
- *public boolean remove(Object x)*: removes a single instance of x from the collection; return true if the collection changed.
- *public Iterator$<$ E $>$ iterator()*: returns an iterator that steps through elements in the collection
- *public Object[] toArray()*: returns a new array containing all the elements of this collection
- *public $<$ T $>$ T[] toArray(T[] dest)*: returns an array containing all the elements of this collection; uses dest as that array if it can.

Java.util.Collection<E>

Bulk operations

- public boolean containsAll(Collection <?> c): Returns true if this collection contains all of the elements in the specified collection.
- public boolean addAll(Collection <? extends E> c): Adds all of the elements in the specified collection to this collection
- public boolean removeAll(Collection <? > c): Removes all of the elements in the specified collection to this collection
- public boolean retainAll(Collection <? > c): Retains only the elements in this collection that are contained in the specified collection
- public void clear(): Clear all elements

Example code: Create a set

see SetToList.java

```java
String[] a = {"ClubsA","SpadeK","HeartsQ","DiamondJ",
  "Clubs10","Spade9","Hearts8","Diamond7",
  "Clubs6","Spade5","Hearts4","Diamond3",
  "Clubs6","Spade4", "Hearts6"};// broken: duplicates

// create a set from the array
// note that duplicates will be removed
Set<String>tmp = new LinkedHashSet<String>(Arrays.asList(a));
System.out.println("Deck: " + tmp);

// a new set can be created from the tmp set
List<String> deck = new ArrayList<String>(tmp);
```

Example code: bulk operators

see MoreSetOps.java

```
String [] d1 = {"one", "two", "three", "four"};
String [] d2 = {"one", "two", "five", "six", "seven"};

Set<String> s1 = new LinkedHashSet<String>(Arrays.asList(d1));
Set<String> s2 = new LinkedHashSet<String>(Arrays.asList(d2));
Set<String> tmp = new LinkedHashSet<String>();//empty set
..
tmp.addAll(s1); // add all elements of s1
..
tmp.addAll(s2); // add all elements of s2
...
tmp.retainAll(s2); // retainAll of s2 in tmp (union of tmp s2)
..
tmp.removeAll(s1); // removeAll of s1 in tmp (diff tmp s1)
```

Example code: collection algorithms

see ListAlgo.java

```
String[] a = {"ClubsA","SpadeK","HeartsQ","DiamondJ",
  "Clubs10","Spade9","Hearts8","Diamond7",
  "Clubs6","Spade5","Hearts4","Diamond3"};

List<String> deck = new ArrayList<String>(Arrays.asList(a));
System.out.println("Sort the Deck");
Collections.sort(deck);
System.out.println(deck);

System.out.println("\nReverse the deck");
Collections.reverse(deck);
System.out.println(deck);

System.out.println("\nShuffle the deck");
Collections.shuffle(deck);
System.out.println(deck);
```

Algorithmic Challenge

Implement a function:

```
public void showAllCombinations(int [])
```

that would display all possible combinations of the given array.

Set Implementation

The *Set* is abstract and there are few implementations that one can select:

HashSet: Best performing implementation.

- Stored in a hash table
- No guarantee on the order when iterating

TreeSet: Slower than *HashSet*

- Stored in a *Red-Black tree*
- Guarantee on the order when iterating

LinkedHashSet: Keeps order with slightly more cost

- Stored in a Hash table with a linked list going through the elements
- Guarantee on the order when iterating

```
Set<Strings> s1 = new HashSet<String>();
Set<Strings> s2 = new TreeSet<String>();
Set<Strings> s3 = new LinkedHashedSet<Strings>();
```

Traversing Collections

- Using **Iterators**
- Using **for-each** construct
- Using **Aggregate operations**
  - Performed method of iteration in JDK 8 and later
  - Often used in conjunction with lambda expressions

Iterators

```
int [] data = {11, 123, 3, 14, 23, 3, 412, 3, 2};
Set<Integer> set = new LinkedHashSet<Integer>();
for(int i=0; i<data.length; i++) set.add(data[i]);

Iterator<Integer> it = set.iterator();
while(it.hasNext())
  System.out.println(it.next());
```

see IterationEx.java

For-each

```
int [] data = {11, 123, 3, 14, 23, 3, 412, 3, 2};
Set<Integer> set = new LinkedHashSet<Integer>();
for(int i=0; i<data.length; i++) set.add(data[i]);

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

see ForEachSample.java

java.util.Iterator<E>

- *public boolean hasNext()*: return true if the iterator has more elements
- *public E next()*: returns the next element in the iteration or throws an exception
- *public void remove()*:
  - removes the last element which was return by `next`
  - move may be called only once per call to `next`
  - otherwise an exception will be thrown
  - Iteration.remove() is the only safe way to modify a collection during iteration