Collection Framework

Limitations with Traditional Arrays

- Can we change array size dynamically?
- Any problems while inserting new element?
- While deleting any element from group?
- Are the elements of Array automatically in sorted order?
- Is searching is easy with array?

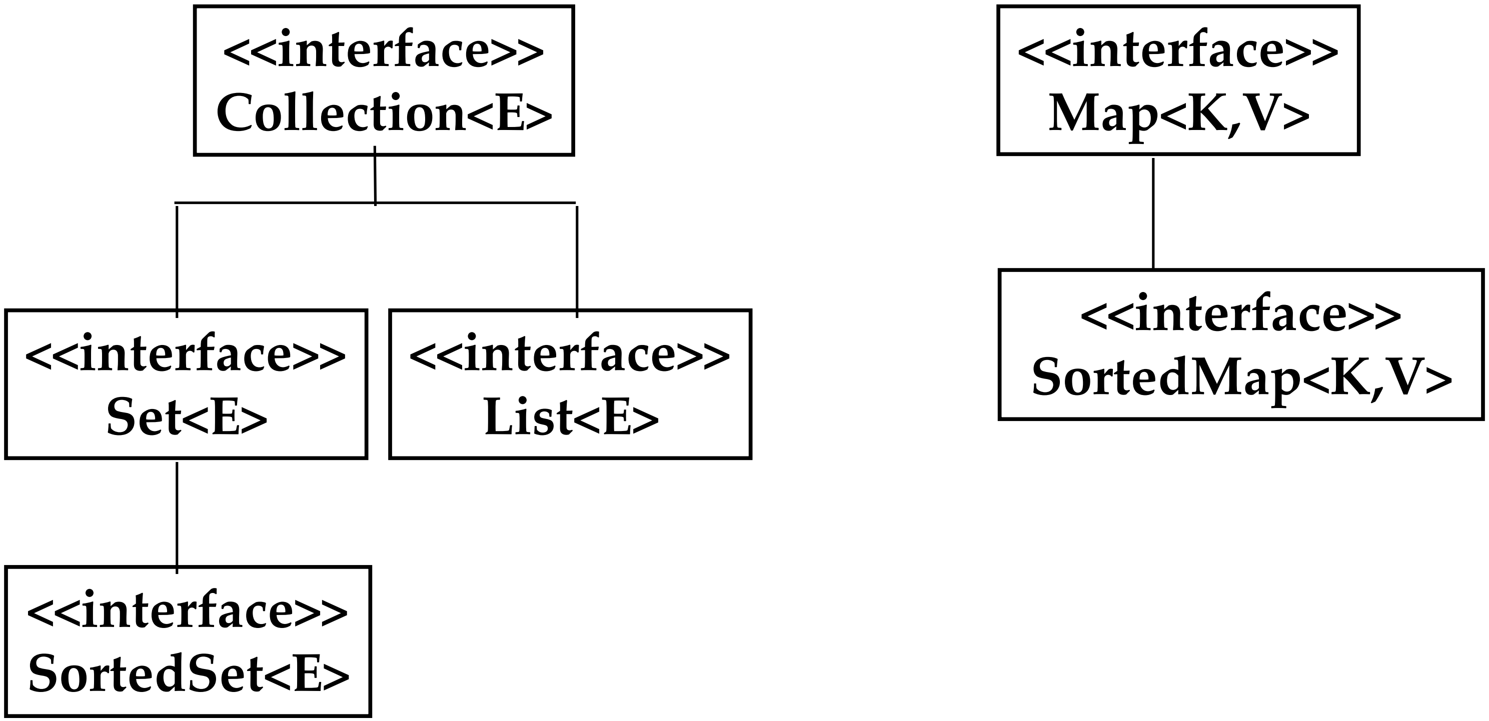
Collection

- Collection—
- A collection is a group of data (object references) handled as a single unit
- What do u mean by framework?
- Do u remember data structures in c?

Data structure Algorithms

- Collection frameworks encapsulates the algorithms
- Programmer don’t have to worry abut the implementation of the various collections types.
- Nevertheless, still collection framework is encourages programmer to extend members of collection to create a specific implementation where one is needed

Collections



Collection framework is in java.util package

How you will implement your ideas

By Extending an abstract class which provides implementation for most of the methods except few methods are left for you as abstract :

AbstractCollection,

AbstractList

AbstractSet

AbstractMap.

By Using one of the concrete classes supplied by the framework.

The Collection<E> interface

- boolean add(E o)
  boolean addAll(Collection c)
  void clear()
  boolean contains(Object o)
  boolean containsAll(Collection c)
  boolean equals(Object o)
  int hashCode()
- boolean isEmpty()
  Iterator<E> iterator()
  boolean remove(Object o)
  boolean removeAll(Collection c)
  boolean retainAll(Collection c)
  int size()
- Object[] toArray()
  Type[] toArray(Type[] a)

- There is no direct concrete implementation class for Collection interface but exists for sub interface

- This interface supports the most basic and general operations and queries that can be performed on a group of objects.

Iterator

- An iterator over a collection.
- Iterator takes the place of Enumeration in the Java collections framework.
- Iterators allow the caller to remove elements from the underlying collection during the iteration
- hasNext():boolean
- next(): E
- remove():void

List interface

- a list is a collection in which duplicate elements are allowed, and where the order is significant

`<a,b,c>`, `<c,a,b>` and `<a,b,b,c>` are different lists

- the List interface inherits from Collection, but changes the semantics of some methods, and adds new methods. The list starts its index at 0.

The List interface

- Additional methods
- - Dealing with position-oriented operations:

```
public void add(int index, E element)
public boolean addAll(int index, Collection c)
public E get(int index)
public int indexOf(Object element)
public int lastIndexOf(Object element)
public Object remove(int index)
public Object set(int index, E element)
```

List implementations

The collections framework contains two concrete implementations to it: ArrayList and LinkedList.

How to select which class will solve your problem?

ArrayList

- ArrayList is recommended when you're adding and removing elements only to the end of the collection satisfies you and when you wish to access elements using their indices.
- It is less time-consuming than LinkedList is.
- The ArrayList provides a collection backed by an array.

LinkedList

- LinkedList is best when add and remove operations happen anywhere, not only at the end.
- But LinkedList's added flexibility comes at an added cost -- it results in much slower indexed operations.

Using List

```
import java.util.*;
class ListDemo
{
  public static void main(String args[]) {
    List<Emp> list = new ArrayList<Emp>();
for (int i=0;i<10;i++)
        list.add(new Emp(i));
  System.out.println(“Third Employee from the list: " +
      (list.get(2).getEmpId());
Iterator<Emp> it = list.iterator();
    while(it.hasNext())
      System.out.println("All the Employees: "+
       (it.next().getEmpId());
 }
}
```

List: Positional access

- E get(int index);
- E set(int index, E element);
- void add(int index, E element);
- Object remove(int index);
- boolean addAll(int index, Collection c);
- These operations are more efficient with the ArrayList implementation

List: Searching

- int indexOf(Object o);
- int lastIndexOf(Object o);

Interface List: Iteration

- Iterators specific to Lists:
- ListIterator<E> listIterator( ) – Places cursor before 1st element
- ListIterator<E> listIterator(int index) –Places cursor before specifed index.

  starts at the position indicated (0 is first element)
- Inherited methods:

  boolean hasNext( );

  E next( );

  void remove( );
- Additional methods:

  boolean hasPrevious()

  E previous()

  int previousIndex()

  int nextIndex()

List: Iterating backwards

- boolean hasPrevious( );
- E previous( );
- int nextIndex( );
- int previousIndex( );

List: More operations

- void add(E o);

  Inserts an object at the cursor position

- E set(E o);   // Optional

  Replace the current element; return the old one

- E remove(int index); // Optional

  Remove and return the element at that position

Queue and Stack

- Implement Stack and Queue using
- Linkedlist

Set

- A collection that contains no duplicate elements
- Set will not maintain any order for elements
- While adding new element it is using Object’s equals() and hashCode() method to check , if such element is there or not in set

Set implementations

- There are three concrete Set implementations that are part of the Collection Framework:
- HashSet, TreeSet, and LinkedHashSet.
- How to select a class which will fulfill your requirement?

HashSet and TreeSet, LinkedHashSet

- You use HashSet, which maintains its collection in an unordered manner.
- If this doesn't suit your needs, you can use TreeSet.
- A TreeSet keeps the elements in the collection in sorted order
- While HashSet has an undefined order for its elements, LinkedHashSet supports iterating through its elements in the order they were inserted.
- Understand that the additional features provided by TreeSet and LinkedHashSet add to the runtime costs.

Map<K,V>

- An object that maps keys to values.
- A map cannot contain duplicate keys.
- each key can map to at most one value.
  - The Data is stored in pairs of objects: key and value
    - Every value object has a key object attached to it.
    - The hash code of key determines where will the pair be stored in the Map.
    - You can only retrieve a value object through its key.
- The Map interface is not extending the Collection interface.
- Map consists of Entries or Mappings
- Entry or mapping consists of key & value type of references.

Map's Methods

- Methods that deal with adding and removing of key-value pairs:

```
public Object put(K key, V value)
public V remove(Object key)
public void putAll(Map<K,V> mapping)
public void clear()
```

Map's Methods

- **Methods that allow you to query the Map's content:**

```
public V get(Object key)
public boolean containsKey(Object key)
public boolean containsValue(Object value)
public int size()
public boolean isEmpty()
```

Map Implementations

- Collection framework gives two classes
  - HashMap and TreeMap
- How to select which will fulfill your requirement

HashMap , TreeMap

- By default, choose HashMap, it serves the most needs.
- TreeMap implementation will maintain the keys of the map in a sorted order.
- it's better to simply keep everything in a HashMap while adding, and create a TreeMap at the end:

-
  ```
  Map <String,String> map = new HashMap<String,String>();
  // Add and remove elements from unsorted map
  map.put("Foo", "Bar");
  map.put("Bar", "Foo");
  map.remove("Foo");
  map.put("Foo", "Baz");
  // Then sort before displaying elements
  // in sorted order
  map = new TreeMap(map);
  ```

Historical Implementations

- List:
- Vector,
- Stack
- Map:
- Hashtable, Properties

What is the Difference?

- The main difference between the historical members we discussed and the new ones is that the old ones are synchronized.
  - -We can ensure about data integrity.
- Synchronization is very resource-consuming, so it is preferable not to automatically synchronize every data structure.