Java's Collection Framework: Examples

Using Set

```
Set set = new HashSet();                    // instantiate a concrete set
set.add(obj);                               // insert an elements
int n = set.size();                         // get size
if (set.contains(obj)) {...}                // check membership

                                            // iterate through the set
Iterator iter = set.iterator();
while (iter.hasNext()) {
  Object e = iter.next();
// ... }
```

Modify the code in the previous slide to

- Given a ArrayList class object “al”. Add all the elements in it to the set.
- The Set object will not allow duplicate elements to be entered
  - Give code which will output a Map: with keys as the unique elements and values as a List of positions where the element occurred in al.
  - Use an iterator to perform the above operations.

Using `Map`

```
Map map = new HashMap(); map.put(key, val);
                                        // insert a key-value pair
                                // get the value associated with key
Object val = map.get(key);
map.remove(key);                        // remove a key-value pair
// ...
if (map.containsValue(val)) { ... }
if (map.containsKey(key)) { ... }

Set keys = map.keySet();                // get the set of keys

                                // iterate through the set of keys
Iterator iter = keys.iterator();
while (iter.hasNext()) {
  Key key = (Key) iter.next();
  // ...}
```

Map views

- Set<K> keySet()
  - Returns a set view of the keys contained in this map.
- Collection<V> values()
  - Returns a collection view of the values contained in this map
  - Can’t be a set—keys must be unique, but values may be repeated

`Map` views

- Set<Map.Entry<K, V>> entrySet()
  - Returns a set view of the mappings contained in this map.
- A view is *dynamic access* into the Map
  - If you change the Map, the view changes
  - If you change the view, the Map changes
- The Map interface does not provide any Iterators
  - However, there are iterators for the above Sets and Collections

```
import java.util.HashMap;
import java.util.Set;

public class HashMapEntrySet1 {
    public static void main(String[] args) {
   //Creating an object of HashMap class
   HashMap<String,Integer> map = new HashMap<String,Integer>(6);

   //Putting key-value pairs inside map
   map.put("Java", 1);
   map.put("is", 2);
   map.put("the", 3);
   map.put("best", 4);
   map.put("programming", 5);
   map.put("language", 6);

   //Creating a Set
   Set set = map.entrySet();

   //Displaying all entries in Set
   System.out.println(set);
}
}
```

```
[the=3, Java=1, is=2, best=4, language=6, programming=5]
```

Using `Vector`

```
Vector v = new Vector(3, 2);          // initial size is 3, increment is 2
   System.out.println("Initial size: " + v.size());
   System.out.println("Initial capacity: " + v.capacity());

   v.addElement(new Integer(1)); …..
   System.out.println("Capacity after four additions: " + v.capacity());
   v.addElement(new Double(5.45));
   System.out.println("Current capacity: " + v.capacity());
   v.addElement(new Double(6.08));  ….

  System.out.println("First element: " + (Integer)v.firstElement());
  System.out.println("Last element: " + (Integer)v.lastElement());

  if(v.contains(new Integer(3)))
      System.out.println("Vector contains 3.");}
```

Using `ListIterator`

For collections that implement List, you can also obtain an iterator by calling ListIterator which can traverse the list in either direction

```
ArrayList al = new ArrayList();

ListIterator litr = al.listIterator();
while(litr.hasNext()) {
     Object element = litr.next();  …. }

// Now, display the list backwards
System.out.print("Modified list backwards: ");
while(litr.hasPrevious()) {
    Object element = litr.previous();
    System.out.print(element + " "); }
```

Ordering and Sorting

There are two ways to define orders on objects.

•Each class can define a *natural order* among its instances by implementing the `Comparable` interface.

```
int compareTo(Object o)
```

•Arbitrary orders among different objects can be defined by *comparators*, classes that implement the `Comparator` interface.

```
int compare(Object o1, Object o2)
```

This method returns zero if the objects are equal. It returns a positive value if o1 is greater than o2. Otherwise, a negative value is returned.

User-Defined Order

Reverse alphabetical order of strings

```
public class StringComparator
     implements Comparator  {
  public int compare(Object o1, Object o2)
{
    if (o1 != null &&
        o2 != null &&
        o1 instanceof String &&
        o2 instanceof String) {
      String s1 = (String) o1;
      String s2 = (String) o2;
      return - (s1.compareTo(s2));
    } else {
      return 0;
    }
  }
}
```

```
import java.util.*;

public class SetIteratorExample
{
   public static void main(String[] args)
   {Set<Integer> set = new HashSet<>();
     for(int i=0; i<5; i++)
             set.add(i);
     Iterator iterator = set.iterator();
             while(iterator.hasNext()){
                         int i = (int) iterator.next();
                         System.out.print(i + ", ");
             }
             //modification of set using iterator
             iterator = set.iterator();
             while(iterator.hasNext()){
                         int x = (int) iterator.next();
                         if(x%2 ==0)
                                     iterator.remove();
             }
             System.out.println(set);
//changing set structure while iterating
iterator = set.iterator();
while(iterator.hasNext()){
 //ConcurrentModification
//Exception here
int x = (int) iterator.next();
if(x==1) set.add(10);
}
}
}
```