Collections

- ‘Augmenting’ the facilities of the java.util package

‘Our main design goal was to produce an API that was reasonably small, both in size, and (more importantly) in “conceptual weight.” It was critical that the new functionality not seem alien to current Java programmers; it had to augment current facilities, rather than replacing them.’

Collections

- why?
  - old java.util drawbacks…
    - limited
    - irregular
- advantages
  - interoperability between unrelated APIs
  - reduces the effort required to learn APIs
  - reduces the effort required to design and implement APIs
  - further fosters software reuse

Collections

- API consists of:
  - Collection hierarchy
    - four core interfaces: collection, set, list, map
  - concrete implementations
    - HashSet, TreeMap, LinkedList, etc.
  - anonymous implementations
    - static factory methods that return Collections
  - abstract implementations
    - “shape defining” implementations guiding rest of APIs
  - infrastructure
    - iterators, ordering comparisons, exceptions
  - algorithms
    - sort, binarySearch, min, max

Collections

		Implementations			
		Hash Table	**Resizable Array**	**Balanced Tree**	**Linked List**
Interfaces	**Set**	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

Collections

- Vector and HashTable remain
  - synchronized implementations of ArrayList and HashMap
- array utilities also “thrown in”
  - searching, sorting, Arrays.asList method…
- iterators
  - similar to enumerators but
    - bi-directional, allows removal from collection, etc.
    - *“designed to eventually supplant the use of Enumeration.”*

Collections

- UnsupportedOperationException
  - attempt to do something that is statically plausible but which is not supported by the collection object at runtime

```
import java.util.*;
public class Coll
  {
  public static void main (String [] args)
    {
    String [] things = { "sugar", "spice", "\'all things nice\'" };
    List thingsList = Arrays.asList (things);
    ListIterator i = thingsList.listIterator ();
    System.out.print ("Forward:");
    while (i.hasNext ())
      {
      String s = (String) i.next ();
      if ("sugar".equals (s))
        i.remove ();
      System.out.print (" " + (String) i.next ());
      }
    System.out.println ();
    System.out.print ("Backward:");
    while (i.hasPrevious ())
      System.out.print (" " + (String) i.previous ());
    System.out.println ();
    }
  }
```

```
>java Coll
Forward:Exception in thread "main" java.lang.UnsupportedOperationException
        at java.util.AbstractList.remove(AbstractList.java:152)
        at java.util.AbstractList$Itr.remove(AbstractList.java:376)
        at Coll.main(Compiled Code)
```

Collections

- WeakHashMap
  - A hashtable-based Map with *weak keys*:
    - *"an entry will automatically be removed when its key is no longer in ordinary use…the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector…may behave as though an unknown thread is silently removing entries."*
  - useful for keys that may 'vanish' during execution
    - cleanup becomes automatic…
- all collection implementations are unsynchronized
  - need to use special synchronizing wrappers
  - also provided: "fail-fast" iterators which can detect concurrent modification to the backing collection and throw ConcurrentModificationException

Collections

- Collections.synchronizedList method
  - returns a thread-safe List backed by a specified List
  - *"It is imperative that the user manually synchronize on the returned List when iterating over it … Failure to follow this advice may result in non-deterministic behavior."*

```
List list = Collections.synchronizedList (new ArrayList());
…
synchronized (list)
  {
  Iterator i = list.iterator (); // Must be in synchronized block
  while (i.hasNext ())
    foo (i.next ());
 }
```

  - also Collections.synchronized{Map, SortedSet}, etc.