**Collections Class**

- This class consists exclusively of static methods that operate on or return collections
- It contains
  - Polymorphic algorithms that operate on collections
  - Wrappers which return a new collection backed by a specified collection
  - A few other odds and ends.

**Algorithms**

- All of the algorithms, provided by the `Collections` class, take the form of static methods
  - Most of the algorithms operate on `List` objects, but a couple of them (`max` and `min`) operate on arbitrary `Collection` objects

**Sorting**

- The sort operation uses a slightly optimized merge sort algorithm
  - Fast: This algorithm is guaranteed to run in $n \log(n)$ time, and runs substantially faster on nearly sorted lists.
  - Stable: That is to say, it doesn't reorder equal elements.
Arrays

• It is too bad that arrays are not collections
  – You loose all of the power provided by the collection framework
• The class Arrays contains
  – various methods for manipulating arrays (such as sorting and searching)
  – It also contains methods that allows arrays to be viewed as lists.
Other Algorithms

- Other algorithms provided by the Collections class include
  - Shuffling
  - Data manipulation
    - reverse()
    - fill()
    - copy()
  - Searching
  - Finding extreme values
    - max()
    - min()

Wrapper Implementations

- Wrapper implementations add some functionality on top of what a collection offer
  - Synchronization
  - Unmodifiable
- Wrappers simply delegate all of their real work to a specified collection

Unmodifiable wrappers

- The unmodifiable wrappers, rather than adding functionality to the wrapped collection, take functionality away.
  - Any attempt to modify the collection generates an UnsupportedOperationException
- The unmodifiable wrappers have two main uses:
  - To make a collection immutable once it has been built.
  - To allow "second-class citizens" read-only access to your data structures. You keep a reference to the backing collection, but hand out a reference to the wrapper.
Unmodifiable wrappers

```java
public static Collection unmodifiableCollection(Collection c);
public static Set unmodifiableSet(Set s);
public static List unmodifiableList(List list);
public static Map unmodifiableMap(Map m);
public static SortedSet unmodifiableSortedSet(SortedSet s);
public static SortedMap unmodifiableSortedMap(SortedMap m);
```

What About User Objects?

- The Collections framework will work with any Java class
- You need to be sure you have defined
  - equals()
  - hashCode()
  - compareTo()
- Don’t use mutable objects for keys in a Map

hashCode()

- hashCode() returns distinct integers for distinct objects.
  - If two objects are equal according to the equals() method, then the hashCode() method on each of the two objects must produce the same integer result.
  - When hashCode() is invoked on the same object more than once, it must return the same integer, provided no information used in equals comparisons has been modified.
  - It is not required that if two objects are unequal according to equals() that hashCode() must return distinct integer values.
Interface Comparable

- This ordering is referred to as the class's natural ordering, and the class's compareTo() method is referred to as its natural comparison method.
- A class's natural ordering is said to be consistent with equals if and only if (el.compareTo(e2)==0) has the same boolean value as: el.equals(e2) for every el and e2 of class C.

```
public class Name implements Comparable<Name> {
    private String first;
    private String last;
    public Name( String firstName, String lastName ) {
        first = firstName;
        last = lastName;
    }
    public String getFirst() {
        return first;
    }
    public String getLast() {
        return last;
    }
    public boolean equals( Object o ) {
        boolean retval = false;
        if (o !=null && o instanceof Name ) {
            Name n = ( Name )o;
            retval = n.getFirst().equals( first ) &&
                n.getLast().equals( last );
        }
        return retval;
    }
    public int hashCode() {
        return first.hashCode() + last.hashCode();
    }
    public String toString() {
        return first + " " + last;
    }
}
```
Name

public int compareTo(Name n) {
    int retval;
    retval = last.compareTo(n.getLast());
    if (retval == 0) {
        retval = first.compareTo(n.getFirst());
    }
    return retval;
} //Name