Excerpts from DisplayableCollections package

```
public abstract class AbstractDisplayableMap<K, V> extends Observable
                                                implements DisplayableMap<K, V>,
                                                           Serializable {
    protected Map<K,V> encapsulatedMap;
...
    protected AbstractDisplayableMap(Map<K,V> encapsulatedMap) {
        this.encapsulatedMap = encapsulatedMap;
    }
...
    public boolean containsKey(Object key) {
        return encapsulatedMap.containsKey(key);
    }
...
    public V get(Object key) {
        return (V) encapsulatedMap.get(key);
    }
...
    public V put(K key, V value) {
        if (encapsulatedMap.containsKey(key)) {
            V oldValue =  encapsulatedMap.put(key, value);
            changed(new Notification<K,V>
                                (ChangeNotification.ChangeType.ELEMENT_MODIFIED,
                                 key, value));
            return oldValue;
        }
        else {
            encapsulatedMap.put(key, value);
            changed(new Notification<K,V>
                                (ChangeNotification.ChangeType.ELEMENT_ADDED,
                                 key, value));
            return null;
        }
    }
....
    public V remove(Object key) {
        if (encapsulatedMap.containsKey(key)) {
            V oldValue = encapsulatedMap.remove(key);
            changed(new Notification<K,V>
                                (ChangeNotification.ChangeType.ELEMENT_REMOVED,
                                 key, oldValue));
            return oldValue;
        }
        else
            return null;
    }
...
    /** Notify observers concerning a change
     *  @param change the change
     */
    protected void changed(Notification<K,V> change) {
        setChanged();
        notifyObservers(change);
...
    }
}
```

```
abstract class ListModelAdapter extends AbstractListModel
                                       implements ListModel, Observer {

    /** Subclass used for sets and lists - this object maintains a Vector of
     *  collection elements
     */
    static abstract class ForCollection<E> extends ListModelAdapter {
...
        // Methods required by the ListModel interface

        public Object getElementAt(int index) {
            return elements.elementAt(index);
        }

        public int getSize() {
            return elements.size();
        }

        void clear() {
            elements.clear();
        }

        Vector<E> elements;
    }

---- A similar subclass is used for maps, with a vector of pairs
...
    // Method required by the Observer interface

    public void update(Observable observable, Object arg) {

        ChangeNotification change = (ChangeNotification) arg;

        int index;

        switch(change.getType()) {

            case CLEARED:
                int oldSize = getSize();
                clear();
                fireIntervalRemoved(this, 0, oldSize - 1);
                break;
            case ELEMENT_ADDED:
                add(change);
                break;
            case ELEMENT_REMOVED:
                remove(change);
                break;
            case ELEMENT_MODIFIED:
                replace(change);
                break;
        }
    }
...
--- Abstract methods clear(), add(), remove(), and replace() are implemented
--- appropriately for the different kinds of collection / map - e.g. the
--- concrete subclass used for a TreeSet keeps the elements in alphabetical
--- order.
```