hierarchy and *implements* the methods of another. In most cases, this is quite a powerful solution. Thus, by using the *implements* keyword, we can have the class take on the methods and appearance of being a class of either type.

The Object Adapter

In the object adapter approach, shown in Figure 9.2, we create a class that *contains* a JList class but which implements the methods of the previous awtList interface. Since the outer container for a JList is not the list element at all but the JScrollPane that encloses it, we are really adding methods to a subclass of JScrollPane that emulate the methods of the List class. These methods are in the interface awtList.

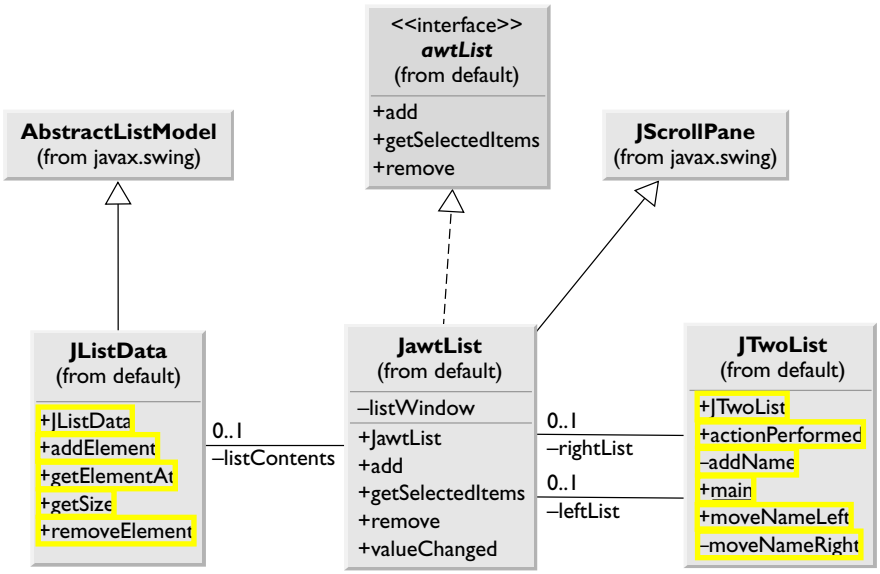

Figure 9.2 An object adapter approach to the list adapter.

So, our basic JawtList class looks as follows:

```
public class JawtList extends JScrollPane
implements ListSelectionListener, awtList {
    private JList listWindow;
    private JListData listContents;

    public JawtList (int rows) {
        listContents = new JListData();
        listWindow   = new JList(listContents);
        listWindow.setPrototypeCellValue("Abcdefg Hijkmnop");
        getViewport().add(listWindow);
    }
public class JawtList extends JScrollPane
```