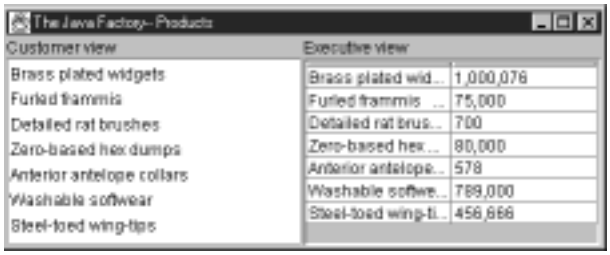

Figure 10.1 Two displays of the same information using a Bridge pattern.

Now, we want to define a single, simple interface that remains the same regardless of the type and complexity of the actual implementation classes. We'll start by defining an abstract Bridger class.

```
public abstract class Bridger extends JPanel {
    public abstract void addData(Vector v);
}
```

This class is so simple that it receives only a Vector of data and passes it on to the display classes.

On the other side of the bridge are the implementation classes, which usually have a more elaborate and somewhat lower-level interface. Here we'll have them add the data lines to the display one at a time.

```
public interface visList {
    public void addLine(String s);
    public void removeLine(int num);
}
```

Implicit in the definition of these classes is some mechanism for determining which part of each string is the name of the product and which part is the quantity shipped. In this simple example, we separate the quantity from the name with two dashes and parse these apart within the list classes.

The Bridge between the interface on the left and the implementation on the right is the listBridge class, which instantiates one or the other of the list display classes. Note that it extends the Bridger class for the use of the application program.

```
public class listBridge extends Bridger {
    protected visList list;

    public listBridge(visList jlist) {
        setLayout(new BorderLayout());
        list = jlist;
        add("Center", (JComponent)list);
    }
    public void addData(Vector v) {
        for (int i = 0; i < v.size(); i++) {
             String s = (String)v.elementAt (i);
```