The Command Pattern

Now, while it is advantageous to encapsulate the action in a Command object, binding that object into the element that causes the action (such as the menu item or button) is not exactly what the Command pattern is about. Instead, the Command object should be separate from the invoking client so that we can vary the invoking program and the details of the command action separately. Rather than having the command be part of the menu or button, we make the Menu and Button classes *containers* for a Command object that exists separately. We thus make UI elements that implement a CommandHolder interface.

```
public interface CommandHolder (
    public void setCommand(Command comd);
    public Command getCommand();
}
```

This interface indicates that there is a method to put a Command object into the invoking object and a method to fetch that Command object and call its *Execute* method.

Then we create the cmdMenu class, which implements this interface.

```
    public class cmdMenu extends JMenuItem
                implements CommandHolder {
    protected Command menuCommand;       //internal copies
    protected JFrame frame;

    public cmdMenu(String name, JFrame frm) {
        super(name);           //menu string
        frame = frm;           //containing frame
    }
    public void setCommand(Command comd) {
        menuCommand = comd;    //save the command
    }
    public Command getCommand() {
        return menuCommand;    //return the command
    }
}
```

This actually simplifies our program—we don't have to create a separate menu class for each action that we want to carry out. We just create instances of the menu and pass them different Command objects.

```
    mnuOpen = new cmdMenu("Open...", this);
    mnuFile.add(mnuOpen);
    mnuOpen.setCommand (new fileCommand(this));
```