To implement this approach, we create several little classes, each of which implements the ActionListener interface.

```
    class btnRed implements ActionListener    {
        public void actionPerformed(ActionEvent e)    {
            p.setBackground(Color.red);
        }
    }
    //--------------
    class fileExit implements ActionListener    {
        public void actionPerformed(ActionEvent e {
            System.exit(0);
        }
    }
}
```

Then we register them as listeners in the usual way.

```
    mnuOpen.addActionListener(new fileOpen());
    mnuExit.addActionListener(new fileExit());
    buttonRed.addActionListener(new btnRed());
```

Here, we made these to be inner classes. However, they also could be external with arguments passed in, as we did previously.

Consequences of the Command Pattern

The Command pattern's main disadvantage is the proliferation of little classes that either clutter up the main class if they are inner classes or clutter up the program namespace if they are outer classes.

Even when we put all of the actionPerformed events in a single basket, we usually call little private methods to carry out the actual functions. It turns out that these private methods are just about as long as out little inner classes, so often there is little difference in complexity between the inner class and outer class approaches.

Anonymous Inner Classes

We can reduce the clutter of the namespace by creating unnamed inner classes. We do this by declaring an instance of a class where we need it. For example,