The program consists of the File Menu object with the mnuOpen and mnuExit MenuItems added to it. It also contains one button called btnRed. Clicking any of these causes an ActionEvent, which generates a call to the *actionPerformed* method such as the following:

```
public void actionPerformed(ActionEvent e)    {
        Object obj = e.getSource();
        if(obj == mnuOpen)
            fileOpen();        //open file
        if (obj == mnuExit)
            exitClicked();     //exit from program
        if (obj == btnRed)
            redClicked();      //turn red
    }
```

Here are the three private methods that we call from the *actionPerformed* method:

```
    private void exitClicked()     {
        System.exit(0);
    }
    //--------------
    private void fileOpen()     {
        FileDialog fDlg = new FileDialog(this, "Open a file",
                                         FileDialog.LOAD);
            fDlg.show();
    }
    //--------------
    private void redClicked()     {
        p.setBackground(Color.red);
    }
```

Now, as long as there are only a few menu items and buttons, this approach works fine, but when there are dozens of menu items and several buttons, the *actionPerformed* code can get pretty unwieldy. This also seems a little inelegant, since, when using an OO language such as Java, we want to avoid a long series of if statements to identify the selected object. Instead, we want to find a way to have each object receive its commands directly.

Command Objects

One way to ensure that every object receives its own commands directly is to use the Command pattern and create individual Command objects. A Command object always has an *Execute* method that is called when an action occurs on that object. Most simply, a Command object implements at least the following interface: