```
public void actionPerformed(ActionEvent e) {
        CommandHolder cmdh = (CommandHolder)e.getSource ();
        Command       cmd  = cmdg.getCommand ();
        u_cmd.add (cmd);     //add to list
        cmd.Execute ();      //and execute
    }
```

Further, the list that we add the Command objects to is maintained inside the Undo Command object so that it can access that list conveniently.

```
public class undoCommand implements Command {
    Vector undoList;

    public undoCommand () {
        undoList = new Vector();        //list of commands to undo
    }
    //--------------
    public void add(Command cmd) {
        if(! (cmd instanceof undoCommand))
            undoList.add(cmd);          //add commands to list
    }
    //--------------
    public void Execute() {
        int index = undoList.size () -1;

        if (index > = 0) {
            //get last command executed
            Command cmd =  (Command)undoList.elementAt (index);
            cmd.unDo  ();               //undo it
            undoList.remove (index);    //and remove from list
        }
    }
    //--------------
    public void unDo() {                //does nothing
    }
}
```

The undoCommand object keeps a list of Commands and not a list of actual data. Each Command object has its *unDo* method called to execute the actual undo operation. Note that since the undoCommand object implements the Command interface, it too must have an *unDo* method. However, the idea of undoing successive unDo operations is a little complex for this simple example program. Consequently, you should note that the *add* method adds all Commands to the list *except* the undoCommand itself, since we have just decided that undoing an unDo command is meaningless.

The Command objects that draw the red and blue lines are derived from the parent drawCommand class. The derived redCommand and blueCommand classes use different colors and start at opposite sides of the window. Each class