THE COMMAND PATTERN IN ACTION: USER INTERFACES

COMMAND + OBSERVER = UI MAGIC

TO ADD A BUTTON WHICH CHANGES
THE CONTENTS OF THE TEXT LABEL..

CREATE THE BUTTON AND ADD IT
TO THE FRAME

```
JButton button = new JButton("Click me");
frame.getContentPane().add(button);
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText("you clicked the button did n't you?");
    }
});
```

COMMAND OBJECT THAT
ENCAPSULATES THE ACTION
TO BE EXECUTED WHEN THE
BUTTON IS CLICKED

REGISTER YOUR EVENT LISTENER
SO THAT IT LISTENS TO THE BUTTON
AND IS NOTIFIED WHEN THE BUTTON
IS CLICKED

THE COMMAND OBJECT IS
EXECUTED WHEN THE EVENT
OCCURS

THE BUTTON IS BOTH THE INVOKER
(COMMAND PATTERN) AND THE
PUBLISHER (OBSERVER PATTERN)

THE BUTTON ACCEPTS ANY COMMAND
OBJECT THAT IMPLEMENTS THE
"ACTIONLISTENER" INTERFACE - WHICH
HAS A SINGLE METHOD

MENUS ARE SET UP THIS WAY TOO

```java
// Create the menu bar
JMenuBar menuBar = new JMenuBar();

// Build the menu
JMenu menu = new JMenu("File");
menuBar.add(menu);

JMenuItem menuItem = new JMenuItem("Save");
menu.add(menuItem);
```

CREATE THE INVOKERS
(THE MENU UI ELEMENTS)

```java
// Create the command object
ActionListener action = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // within the command object, do what's needed with the receiver(s)
        final JFileChooser fc = new JFileChooser();
        fc.setDialogTitle("Save your work");

        if (fc.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
            HTMLWriter.writeToHTML(fc.getSelectedFile().getAbsolutePath(), urlInfoMap.values());
        }
    }
};
// attach the command object to the invoker
menuItem.addActionListener(action);
```

CREATE THE COMMAND OBJECT

INSIDE THE COMMAND OBJECT,
DO WHAT'S NEEDED WITH THE
RECEIVERS

ASSOCIATE THE COMMAND OBJECT
WITH THE INVOKER

THE COMMAND PATTERN SEPARATES
THE EXECUTION OF AN ACTION FROM THAT
ACTION ITSELF

THIS MAKES THE COMMAND PATTERN
THE BASIS OF "UNDO-REDO" FUNCTIONALITY
IN APPLICATIONS

TO IMPLEMENT UNDO IN YOUR APPLICATION -

REPRESENT EVERYTHING THAT CAN POSSIBLY HAPPEN AS A COMMAND OBJECT

(IN MOST UI APPLICATIONS, THIS REPRESENTS VIRTUALLY NO EXTRA WORK, BECAUSE EVERYTHING THAT HAPPENS IS ALREADY A COMMAND OBJECT WIRED UP TO SOME UI ELEMENT)

BUT THERE IS ONE TRICKY BIT - EACH COMMAND OBJECT MUST ALSO KNOW HOW TO UNDO ITSELF

THIS IS INDEED ADDITIONAL WORK - EACH ACTION MUST ALSO IMPLEMENT AN INTERFACE WITH AN UNDO METHOD

WHENEVER ANYTHING HAPPENS, FIND THE ACTION ITEM THAT IS EXECUTED, AND ADD IT TO THE END OF A LIST

(MAINTAINING SUCH A LIST AS A MEMBER VARIABLE OF YOUR UI CLASS IS ALSO PRETTY EASY)

LOGGING

I.E. MAINTAINING A TRACK OF EVERY ACTION THAT HAS OCCURRED IN YOUR APPLICATION

AS WITH UNDO - YOU NEED TO REPRESENT ANYTHING THAT CAN HAPPEN AS A COMMAND OBJECT

ALSO AS WITH UNDO, YOU NEED TO MAINTAIN A LIST AND KEEP ADDING ACTIONS ONTO THEM AS THEY OCCUR

UNLIKE WITH UNDO, YOU DO NOT NEED EACH ACTION TO KNOW HOW TO UNDO ITSELF

BUT YOU PROBABLY NEED YOUR COMMAND OBJECTS TO KNOW HOW TO WRITE THEMSELVES OUT TO A FILE

(AN OBJECT THAT KNOWS HOW TO DO THIS IS SAID TO BE SERIALIZABLE)

(THIS IS BECAUSE LOGGING IS OFTEN NEEDED EXACTLY WHEN AN APPLICATION CRASHES!)