**Command**

**Definition**

Streamlize objects by providing an interface to encapsulate a request and make the interface implemented by subclasses in order to parameterize the clients.

**Where to use & benefits**

* One action can be represented in many ways, like drop-down menu, buttons and popup menu.
* Need a callback function, i.e., register it somewhere to be called later.
* Specify and execute the request at different time
* Need to undo an action by storing its states for later retrieving.
* Decouple the object with its trigger
* Easily to be extensible by not touching the old structure.
* Related patterns include
  + [Composite](http://www.javacamp.org/designPattern/composite.html), which aggregates an object. You may combine it into a composite command pattern. In general, a composite command is an instance of the composite.
  + [Memento](http://www.javacamp.org/designPattern/memento.html), which keeps state of an object. Command supports undo and redo.

**Example**

The simple example of Command pattern is to design a Command interface and with an execute method like this:

public interface Command {

public void execute();

}

Then, design multiple implementation classes and see how powerful the execute() method has been called dynamically.

In order to take advantage of Java built-in interfaces, we will design a window with a drop down menu, button commands and popup menu with command pattern.

As we know, JButton, JMenuItem and JPopupMenu have constructors accept Action type variable. Action interface extends ActionListener, which has the following hierarchy.

public interface EventLister {

...

}

public interface ActionListener extends EventListener {

...

}

public interface Action extends ActionListener {

...

}

There is an abstract class called AbstractAction which implements Action interface. It has the following design.

public abstract class AbstractAction extends Object

implements **Action**, Cloneable, Serializable

We will create several command classes to subclass the AbstractAction class and pass them to the constructors of JButton, JMenuItem and JPopupMenu classes. There is a request method called actionPerformed(), every command classes must implement it in order to make it work. To show the concept, we just design two actions: submit and exit. You may expand such design to your need in your future project.

Such action can be attached to any component, AWT or Swing. The caption, and Icon have been designed as well as tooltips.

class ExitAction extends AbstractAction {

private Component target;

public ExitAction(String name, Icon icon, Component t){

putValue(Action.NAME, name);

putValue(Action.SMALL\_ICON, icon);

putValue(Action.SHORT\_DESCRIPTION, name + " the program");

target = t;

}

**public void actionPerformed(ActionEvent evt)** {

int answer = JOptionPane.showConfirmDialog(target, "Are you sure you want to exit? ", "Confirmation",

JOptionPane.YES\_NO\_OPTION);

if ( answer == JOptionPane.YES\_OPTION) {

System.exit(0);

}

}

}

Similar to the above exit action, the submit action is as follows:

class SubmitAction extends AbstractAction {

private Component target;

public SubmitAction(String name, Icon icon, Component t){

putValue(Action.NAME, name);

putValue(Action.SMALL\_ICON, icon);

putValue(Action.SHORT\_DESCRIPTION, name + " the program");

target = t;

}

**public void actionPerformed(ActionEvent evt)** {

JOptionPane.showMessageDialog(target, "submit action clicked ");

}

}

You can modify the program to add more commands in. These command classes are decoupled from any program. It is very good for maintenance.

The whole workable program is as follows. You can run it to see the powerful command design pattern.

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

class ExitAction extends AbstractAction {

private Component target;

public ExitAction(String name, Icon icon, Component t){

putValue(Action.NAME, name);

putValue(Action.SMALL\_ICON, icon);

putValue(Action.SHORT\_DESCRIPTION, name + " the program");

target = t;

}

public void actionPerformed(ActionEvent evt) {

int answer = JOptionPane.showConfirmDialog(target, "Are you sure you want to exit? ", "Confirmation",

JOptionPane.YES\_NO\_OPTION);

if ( answer == JOptionPane.YES\_OPTION) {

System.exit(0);

}

}

}

class SubmitAction extends AbstractAction {

private Component target;

public SubmitAction(String name, Icon icon, Component t){

putValue(Action.NAME, name);

putValue(Action.SMALL\_ICON, icon);

putValue(Action.SHORT\_DESCRIPTION, name + " the program");

target = t;

}

public void actionPerformed(ActionEvent evt) {

JOptionPane.showMessageDialog(target, "submit action clicked ");

}

}

class Test extends JFrame{

Test() {

**Action ea = new ExitAction("Exit", null, this);**

**Action sa = new SubmitAction("Submit", null, this);**

JMenuBar jbr = new JMenuBar();

JMenu dropmenu= new JMenu("File");

JMenuItem submitmenu = new JMenuItem(**sa**);

JMenuItem exitmenu = new JMenuItem(**ea**);

dropmenu.add(submitmenu);

dropmenu.add(exitmenu);

jbr.add(dropmenu);

setJMenuBar(jbr);

final JPopupMenu pop = new JPopupMenu("PopMenu");

pop.add(**sa**);

pop.add(**ea**);

addMouseListener(new MouseAdapter() {

public void mousePressed(MouseEvent e) {

showPopup(e);

}

public void mouseReleased(MouseEvent e) {

showPopup(e);

}

private void showPopup(MouseEvent e) {

if (e.isPopupTrigger()) {

pop.show(e.getComponent(),

e.getX(), e.getY());

}

}

});

JPanel jp = new JPanel();

JButton subbtn = new JButton(**sa**);

JButton exitbtn = new JButton(**ea**);

jp.add(subbtn);

jp.add(exitbtn);

Container con = getContentPane();

con.add(jp, "South");

setTitle("Command pattern example");

setSize(400,200);

setVisible(true);

}

public static void main(String[] args) {

new Test();

}

}

java Test

A windows pops up.

Pay attention to the action buttons. The instances can be parameterized to JButton, JMenuItem and JPopupMenu constructors. The powerful action design (Java built-in Action interface) makes objects like ExitAction, SubmitAction be used everywhere. Design once, use everywhere.