Report on Document Editor

Submitted By:

Gauri Walke - 04

Praja Bobde - 14

Arpit Kidile – 34

Siddharth Golecchha – 64

TABLE OF CONTENTS

1. Introduction

What is Design Pattern

Document Editor

List of Design Pattern used

2.Composite Design Pattern

Intent

Usage

Structure

Code Segment

Description

1. Façade Design Pattern

Intent

Usage

Structure

Code Segment

Description

4.Command Design Pattern

Intent

Usage

Structure

Code Segment

Description

5.Memento Design Pattern

Intent

Usage

Structure

Code Segment

Description

6.Iterator Design Pattern

Intent

Usage

Structure

Code Segment

Description

7.Flyweight Design Pattern

Intent

Usage

Structure

Code Segment

Description

8.Java Code

9.Screenshots

10.Conclusion

11.References

Introduction

DESIGN PATTERN

Design pattern represent the best practices used by experienced object oriented software developers.Design patterns are solutions togeneral problems faced during software development.Learning these patterns help developers to learn software design in a easy and a faster way.Design pattern provide a standard terminology and are specific to particular scenario.

LEXI

*Lexi* is the name of the sample "document editor" used by the Gang of Four, *Gof* for short, as the introductory example to design patterns.

This book written in 1996 by [**Erich GAMMA, Richard HELM, Ralph JOHNSON, John VLISSIDES**](http://www.felix-colibri.com/papers/design_patterns/the_lexi_editor/the_lexi_editor.html#gof_design_patterns) literaly started the whole pattern movement.

After introducing the Design Pattern idea, the Gof spent about 40 pages presenting a case study which explains how to use 8 typical patterns in the course of designing the "*LEXI* document editor". The next 300 pages then explain each of the 23 Design Patterns, with motivation, diagrams and coding examples.

DOCUMENT EDITOR

Document Editor supports text,text size,image insertion,border generation,border color,words count and other functions like undo and redo.

DESIGN PATTERN USED

1.Composite Design pattern

2.Facade Design pattern

3.Command Design pattern

4.Memento Design pattern

5.Iterator Design pattern

6.Flyweight Design pattern

COMPOSITE DESIGN PATTERN

Composite pattern is a partitioning design pattern and describes a group of objects that is treated the same way as a single instance of the same type of object.

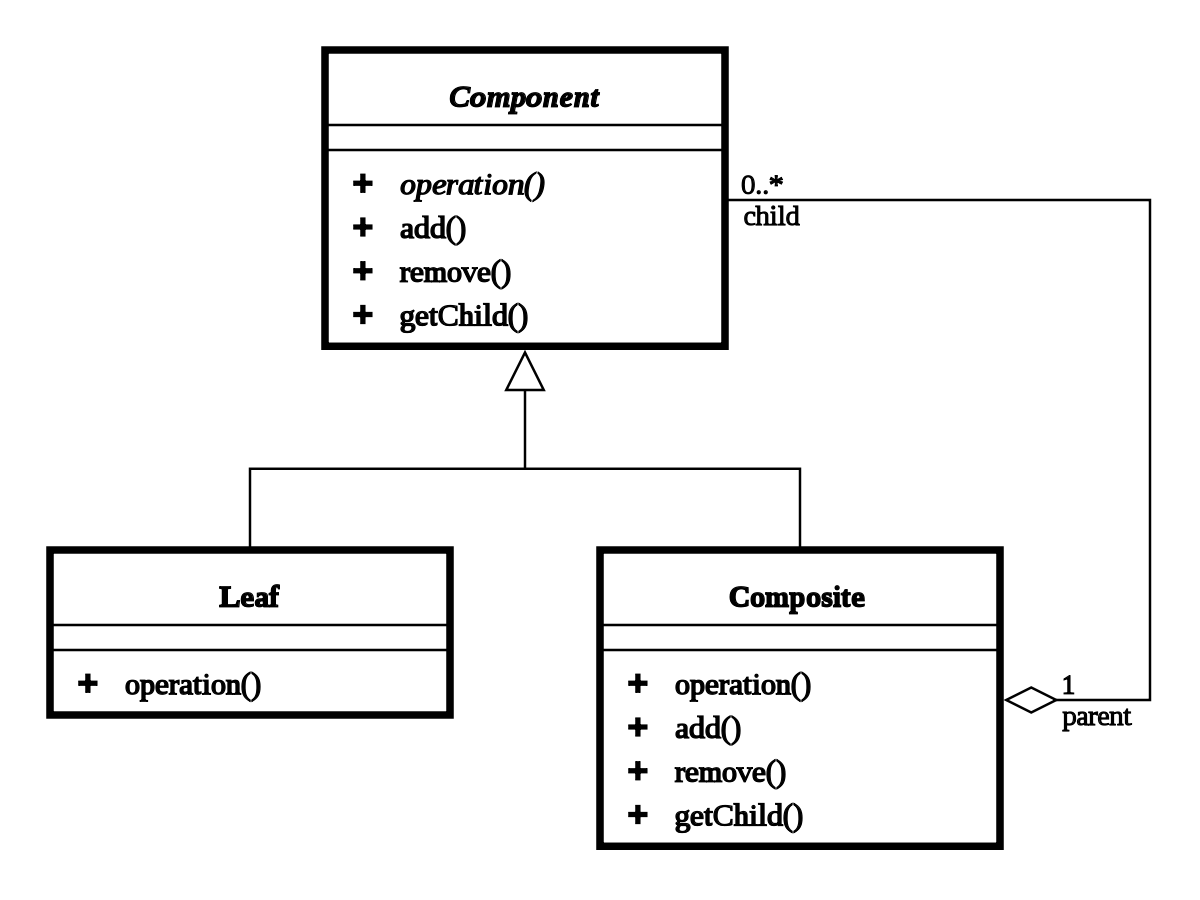
Intent-

The intent of a composite is to “compose” objects into tree structures to represent part-whole hierarchies. It allows you to have a tree structure and ask each node in the tree structure to perform a task.

Usage-

Using this design pattern,we have designed the basic structure of our document editor.It contains JPanel,JButton,JTextField,JcomboBox,Jlabel,etc.All the buttons are in the rightmost part of the editor.

STRUCTURE



CODE SEGMENT

package canvas\_practice;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import sun.font.FontDesignMetrics;

public class Canvas\_Practice extends JComponent implements KeyListener{

static int c=0;

private static final Color m\_tRed = new Color(255, 0, 0, 150);

private static Color m\_tGreen = new Color(0, 255, 0, 150);

private static Color m\_tBlue = new Color(0, 0, 255, 150);

//this done by getting the known information about the fonts

//using the FontMetrics class.

static int width\_of\_character=27;

static int height\_of\_character=46;

static Font monoFont = new Font("Monospaced", Font.BOLD

| Font.ITALIC, 40);

static Canvas\_Practice cp;

static ArrayList<Object>char\_list=new ArrayList<>();

static Image image\_custom=Toolkit.getDefaultToolkit().getImage("C:\\Users\\username\\Desktop\\emoji\_5.png");

static Glyph glyph;

static SystemFacade systemFacade;

static CustomCharacterFlyweightFactory characterFlyweightFactory;

static Image bimage;

static JLabel count\_Words;

static Integer fontSize=40;

static String fontFace="monospaced";

public Canvas\_Practice() {

FontMetrics fm=Toolkit.getDefaultToolkit().getFontMetrics(monoFont);

width\_of\_character=fm.stringWidth("C");

height\_of\_character=fm.getAscent();

glyph=new RowComposite();

characterFlyweightFactory=CustomCharacterFlyweightFactory.getInstance();

systemFacade=new SystemFacade();

}

static Canvas\_Practice getInstanceOf(){

return cp;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

//call rowcomposite and say it to Paint all the components

glyph.draw(g, 0);

}

@Override

public Dimension getPreferredSize() {

return new Dimension(1600, 600);

}

@Override

public Dimension getMinimumSize() {

return new Dimension(500, 500);

}

public boolean containsCharacter(String s) {

return (s == null) ? false : s.matches("[^A-Za-z0-9!@#$%^&\*(),.?\":{}|<> ]");

}

private void Character\_Pressed(java.awt.event.KeyEvent evt) {

// TODO add your handling code here:

if(evt.getKeyCode()==KeyEvent.VK\_BACK\_SPACE){

//do task of removing the component from the List

//remove

systemFacade.removeComponent(glyph);

}

else{

//do task of adding the string

char ch=evt.getKeyChar();

if(!containsCharacter(String.valueOf(ch))){

System.out.println("THis is Character so type it now");

systemFacade.insertComponent(glyph, ch,characterFlyweightFactory);

}

}

System.out.println("Character Pressed : "+evt.getKeyChar()+" Key Code = " +evt.getExtendedKeyCode());

cp.repaint();

}

public static void main(String[] args) throws ClassNotFoundException {

try {

// UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

//UIManager.setLookAndFeel ("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

// TODO code application logic here

JFrame jtf=new JFrame("LEXI");

jtf.getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));

cp=new Canvas\_Practice();

cp.addKeyListener(cp);

cp.setFocusable(true);

cp.setBorder(BorderFactory.createLineBorder(m\_tBlue, 5));

JPanel panel2=new JPanel();

panel2.setLayout(new BoxLayout(panel2, BoxLayout.Y\_AXIS));

jtf.getContentPane().add(cp);

JButton fontChange=new JButton("Font");

fontChange.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFrame f3 = new JFrame();

JLabel size = new JLabel("Enter Size");

JTextField size1 = new JTextField();

f3.setBounds(200,200,300,300);

JButton set = new JButton("Set");

size.setBounds(0,0,100,50);

size1.setBounds(0,50,100,50);

set.setBounds(0,100,100,50);

f3.add(size);

f3.add(size1);

f3.add(set);

f3.setLayout(null);

f3.setResizable(false);

f3.setVisible(true);

set.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if(!"".equals(size1.getText())){

fontSize=Integer.parseInt(size1.getText());

System.out.println("Font SIze: "+fontSize);

systemFacade.changeFontSize(fontSize);

f3.dispose();

cp.repaint();

cp.grabFocus();

}

}

});

cp.repaint();

cp.grabFocus();

}

});

JButton fontface=new JButton("Font Face");

fontface.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFrame f3=new JFrame("Font Styles");

JLabel style=new JLabel("Select font Name");

JButton btn=new JButton("Set Font");

String fonts[] = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();

JComboBox jComboBox=new JComboBox(fonts);

style.setBounds(0,0,150,50);

f3.setBounds(400,400,400,400);

jComboBox.setBounds(0,50,200,50);

btn.setBounds(0, 200, 100,30);

f3.add(style);

f3.add(jComboBox);

f3.add(btn);

btn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

fontFace=jComboBox.getSelectedItem().toString();

if(fontFace!=null){

systemFacade.changeFontFace(fontFace);

cp.repaint();

cp.grabFocus();

}

}

});

}

});

JButton undo=new JButton("Undo");

undo.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

systemFacade.undoTask(glyph);

cp.repaint();

cp.grabFocus();

}

});

JButton redo=new JButton("Redo");

redo.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

systemFacade.redo();

cp.repaint();

cp.grabFocus();

}

});

JButton countGlyph=new JButton("Count Words");

countGlyph.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

systemFacade.countChracters(glyph);

cp.repaint();

cp.grabFocus();

}

});

count\_Words=new JLabel();

count\_Words.setBorder(BorderFactory.createLineBorder(Color.RED));

count\_Words.setBounds(new Rectangle(10,10));

JButton changeBorder=new JButton("Border");

changeBorder.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

Color c1=JColorChooser.showDialog(cp, "Select the Color of the Border", m\_tBlue);

if(c1!=null){

systemFacade.changeBorder(cp,c1);

}

cp.repaint();

cp.grabFocus();

}

});

JButton insertImage=new JButton("Insert Image");

insertImage.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

try {

Image bimage = ImageIO.read(file);

systemFacade.insertImage(glyph, bimage);

cp.repaint();

cp.grabFocus();

} catch (IOException ex) {

//exception handle

}

}

}

});

panel2.add(fontChange);

panel2.add(fontface);

panel2.add(undo);

panel2.add(redo);

panel2.add(insertImage);

panel2.add(changeBorder);

panel2.add(countGlyph);

panel2.add(count\_Words);

jtf.getContentPane().add(panel2);

jtf.setVisible(true);

jtf.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// Dimension dim=Toolkit.getDefaultToolkit().getScreenSize();

//

//

//

// jtf.getContentPane().setSize(dim);

jtf.pack();

} catch (InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException ex) {

Logger.getLogger(Canvas\_Practice.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyPressed(KeyEvent e) {

// System.out.println("Key Pressed");

Character\_Pressed(e);

}

@Override

public void keyReleased(KeyEvent e) {

// System.out.println("Key Released");

}

}

DESCRIPTION-

Basic structure of editor is prepared using JPanel,JTextField,Jbutton,etc.

FAÇADE DESIGN PATTERN

Facade is a part of Gang of Four design pattern and it is categorized under Structural design patterns.

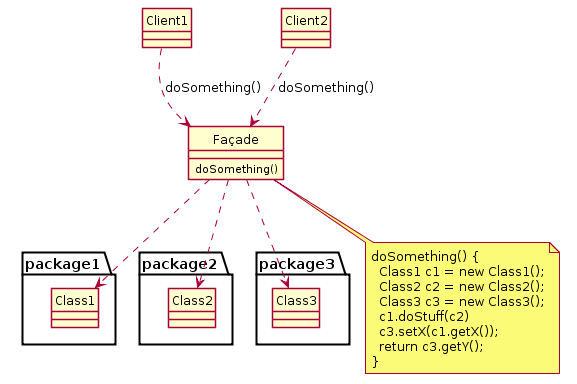
**Intent**

* Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
* Wrap a complicated subsystem with a simpler interface.

USAGE-

It is used to provide different functions in the text editor such as undo,redo,font etc.

STRUCTURE-



CODE SEGMENT-

package canvas\_practice;

import static canvas\_practice.Canvas\_Practice.glyph;

import java.awt.Color;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Image;

import java.awt.Toolkit;

import java.util.Stack;

import javax.swing.BorderFactory;

public class SystemFacade {

Custom\_Rectangle cr;

private final Stack<Command> commandStack;

private final Stack<Memento> mementoStack;

//counter for the undo/redo algorithm

private int undoRedoPointer = -1;

public SystemFacade() {

cr=new Custom\_Rectangle();

commandStack=new Stack<>();

mementoStack=new Stack<>();

}

void insertComponent(Glyph glyph,char ch,CustomCharacterFlyweightFactory cff){

Command c;

Custom\_Character cc=cff.getCustomCharacter(ch);

c=new InsertGlyph(glyph, cc, cr);

String type="";

if(ch==' '){

type="space";

}

else {

type="char";

}

c.execute(type);

undoRedoPointer+=1;

commandStack.push(c);

}

void insertImage(Glyph glyph,Image bimage){

//do addition of iamges here

Custom\_Image ci=new Custom\_Image(bimage,Canvas\_Practice.cp);

glyph.addComponent(ci);

}

void removeComponent(Glyph glyph){

Command c=new RemoveGlyph(glyph);

c.execute("No Need");

commandStack.push(c);

undoRedoPointer+=1;

}

void changeFontSize(Integer fontSize){

String fontFace=Canvas\_Practice.monoFont.getFamily();

Font f=new Font("monospaced", Font.BOLD, fontSize);

FontMetrics fm=Toolkit.getDefaultToolkit().getFontMetrics(f);

Canvas\_Practice.width\_of\_character=fm.stringWidth("C");

Canvas\_Practice.height\_of\_character=fm.getAscent();

Canvas\_Practice.monoFont=f;

}

void changeFontFace(String fontFace){

Integer fontSize=Canvas\_Practice.monoFont.getSize();

Font f=new Font(fontFace, Font.BOLD, fontSize);

FontMetrics fm=Toolkit.getDefaultToolkit().getFontMetrics(f);

Canvas\_Practice.width\_of\_character=fm.stringWidth("C");

Canvas\_Practice.height\_of\_character=fm.getAscent();

Canvas\_Practice.monoFont=f;

}

void undoTask(Glyph glyph){

System.out

.println("undoPointer: " + undoRedoPointer + ", stackSize: " + commandStack.size());

if(undoRedoPointer!=-1){

Command command = commandStack.get(undoRedoPointer);

command.unexecute();

try{

if(undoRedoPointer>=0){

undoRedoPointer-=1;

}

}

catch(ArrayIndexOutOfBoundsException c){

System.out.println("Indexed Out of Bounded");

}

}

}

void redo(){

System.out

.println("undoPointer: " + undoRedoPointer + ", stackSize: " + commandStack.size());

if(undoRedoPointer == commandStack.size() - 1)

{ return; }

undoRedoPointer++;

Command command = commandStack.get(undoRedoPointer);

command.execute("No Params");

}

void countChracters(Glyph component){

int count=0;

System.out.println("-----------------------------------------------------");

CustomIterator itr=component.getIterator();

while(itr.hasNext()){

count+=1;

Glyph next = itr.next();

}

System.out.println("--------------------- Count = "+count+" ------------------------------");

Canvas\_Practice.count\_Words.setText("Count = "+count);

}

void changeBorder(Canvas\_Practice cp,Color c1){

cp.setBorder(BorderFactory.createLineBorder(c1,5));

}

}

DESCRIPTION-

List of functions performed-

1.Font

2.Undo

3.Redo

4.Border

5.Image

6.Count words