Design Patterns

Assignment – 2

Case Study : Designing a Document Editor.

Submitted By –
Rishabh Gupta (54)
Ruchit Bhardwaj (55)
Sai Shashank Mudliar (56)
Sanjeet Kumar (58)
Vinit Aswale (68)
Viraj Dharme (69)
Yogesh Kothari (71)

Table Of Contents

1. Abstract Page 3
2. Introduction Page 3
3. Patterns Used Page 3
    3.1 Composite Design Pattern Page 4
    3.2 Command Design Pattern Page 7
    3.3 Decorator Design Pattern Page 9
    3.4 Bridge Design Pattern Page 11
4. Features Page 13
5. Code Page 17
6. Overall Class Diagram Page 35
7. Output Screenshot Page 35
8. References Page 35

1.0 Abstract

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time. Here in this case study, we use various design patterns to solve the design issues relating to the application of a WYSIWYG document editor.

2.0 Introduction

This application is based in a case study in the design of a “What-You-See-Is-What-You-Get” for (WYSIWUG) document editor called **Lexi**. Various design patterns capture the solutions to design problems in Lexi and applications like it. A WYSIWYG representation of a document occupies a large rectangular area in the center . The document can ideally mix text and graphics freely in a variety of formatting styles. Surrounding the document are the usual pull down menus and the scroll bars.

3.0 Patterns Used

- Composite[8][9][9] Design Pattern
- Command[6][7] Design Pattern
- Bridge[4][5] Design Pattern
- Decorator[2][3] Design Pattern

3.1 Composite Design Pattern

Intent

The formal definition of the Composite[8][9][9] Pattern says that it allows you to compose objects into tree structures to represent part whole hierarchies. Composite[8][9][9] lets clients to treat individual objects and compositions of objects uniformly.

The Composite[8][9][9] Pattern allows us to build structures of objects in the form of trees that contains both composition of objects and individual objects as nodes. Using a composite structure, we can apply the same operations over both composites and individual objects. In other words, in most cases we can ignore the differences between compositions of objects and individual objects.

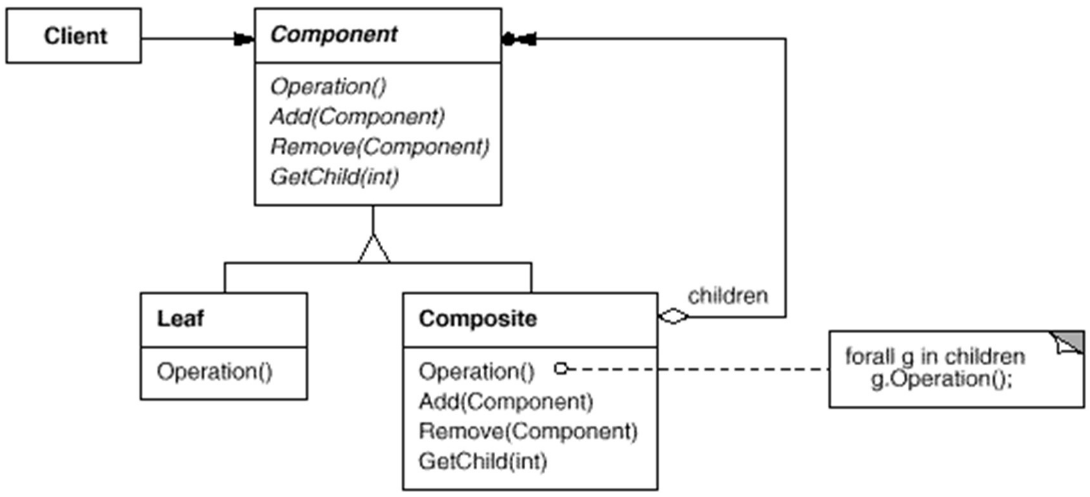
Motivation

- Graphics applications like drawing editors and schematics capture systems let users build complex diagrams out of simple components.
- The users can group components to form larger components, which in turn can be group to form still larger components.
- A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that acts as containers for primitives.
- But there’s a problem with this approach: Code that uses these classes must treat primitives and container objects differently, even if most of the time the user treats them identically. Having to distinguish these objects makes the application more complex.
- The composite pattern describes how to use recursive composition so that clients don’t have to make this distinction.
- The key to the Composite[8][9][9] pattern is an abstract class that represents both primitives and their containers.

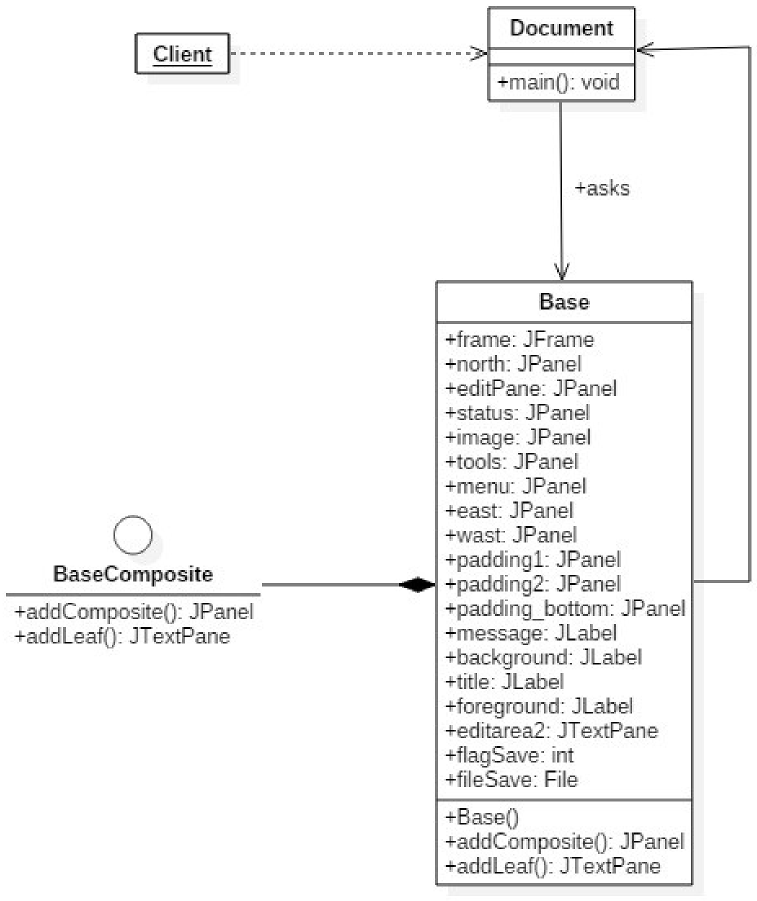
Participants

- Component
- Leaf
- Composite[8][9][9]
- Client

General Structure



Document Editor Specific Structure



Recursive Composition

A common way of representing hierarchically structured information is through a technique called **recursive composition.** It entails building increasingly complex elements out of simpler ones. Recursive composition gives us a way to compose a document out of simple graphical elements. As a first step, we can tile a set of characters and graphics to form a line in the document. Then, multiple lines can be arranged to form a column and multiple columns can form a page.

Working in Code

In the document editor that has been designed, the composite design pattern has been used to construct the user interface of the document editor. The interface has two methods, addComposite[8][9][9]() and addLeaf(). The panels are added in the addComposite[8][9][9]() method since there will be more components to be added in the panel such as buttons and labels whereas the leaf elements such as the TextPane has been added in the addLeaf() method since the TextPane will not be composed of any other component.

3.2 Command Design Pattern

Intent

Encapsulate a request as an object , thereby letting you parametrize clients with different requests, queue or log requests, and support undoable operations.

Command[6][7] pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

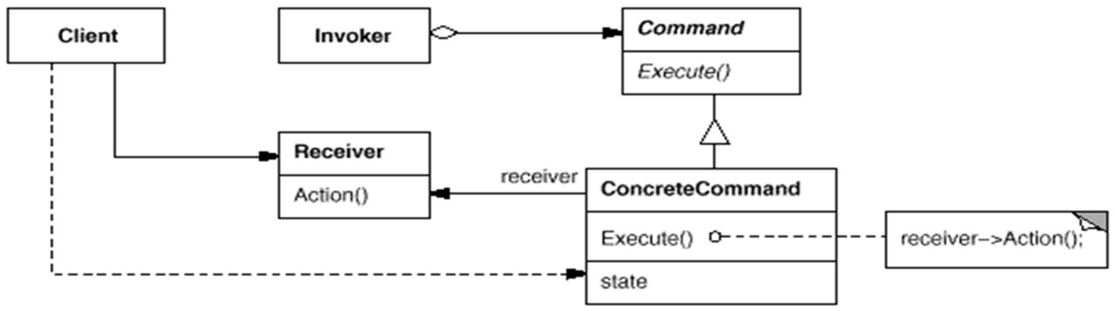
Motivation

- Allows you to store list of code that is executed at a later time or many times.
- It is easy to add new commands because you don't have to change existing classes.
- The command design pattern lets objects make requests of unspecified application objects by turning the request itself into an object. This object can be stored and passed around like other objects. The key to this pattern is an abstract command class which declares an interface for executing operations. In the simplest form this interface includes an abstract Execute operation. Concrete Command[6][7] subclasses specify a receiver-action pair by storing storing receiver as an instance variable and by implementing Execute to invoke the request. The receiver has the knowledge to carry out the request.

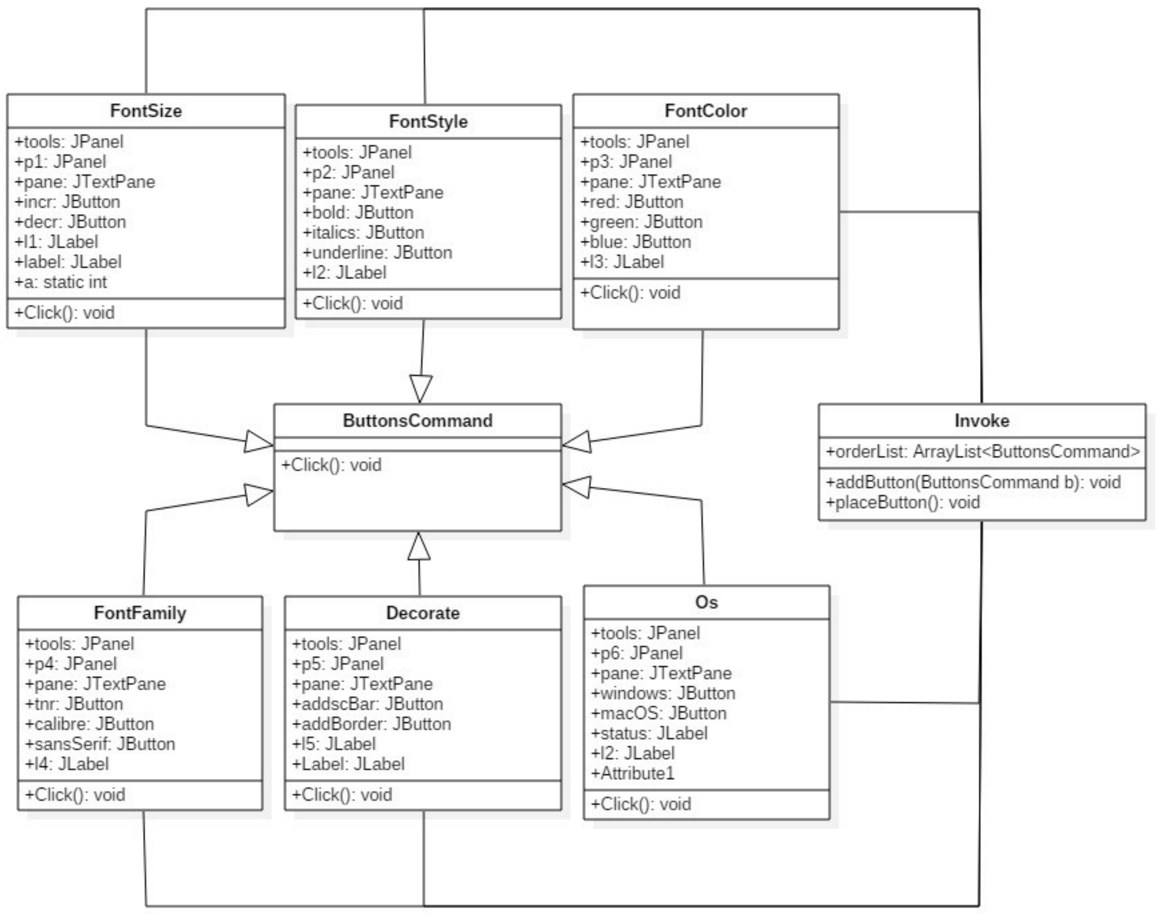
Participants

- Command[6][7]
- ConcreteCommand[6][7]
- Client
- Invoker
- Receiver

General Structure



Documents Editor Specific Structure



Working in Code

In the Document editor Program we have “ButtonsCommand[6][7]” interface having click() method which is implemented by six Concrete subclasses namely FontSize, FontStyle, FontColor, FontFamily, Decorate and OS. We also have a Invoke class which stores all the subclasses of “ButtonsCommand[6][7]” interface in an arraylist. The client invokes the Invoke() class and it’s two methods void addButton(ButtonsCommand[6][7] button) and void placeButton() to interact with the subclasses.

3.3 Decorator Design Pattern

Intent

Decorator[2][3] Design Pattern attaches additional responsibilities to an object dynamically. Decorator[2][3]s provide a flexible alternative to sub-classing for extending functionality.

This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class. This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact. We are demonstrating the use of decorator pattern via following example in which we will decorate a shape with some color without alter shape class.

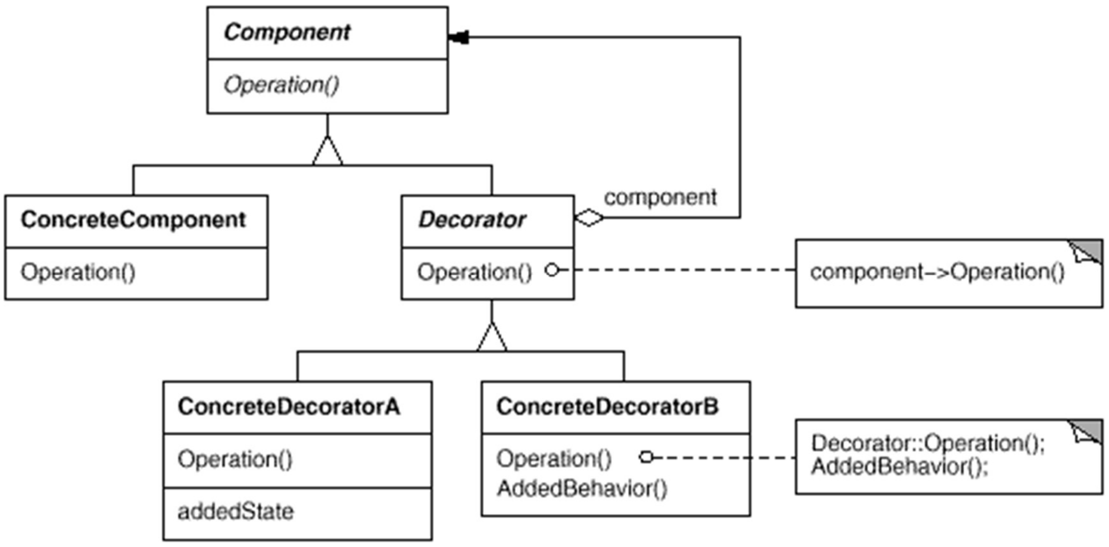
Motivation

- Sometimes we want to add responsibilities to individual objects, not to an entire class. For example, to add properties like borders or behaviors like scrolling to any user interface component.
- One way to add responsibilities is with inheritance. Inheriting a border from another class puts a border around every subclass instance. A more flexible approach is to enclose the component in another object that adds the border. The enclosing object is called a decorator.

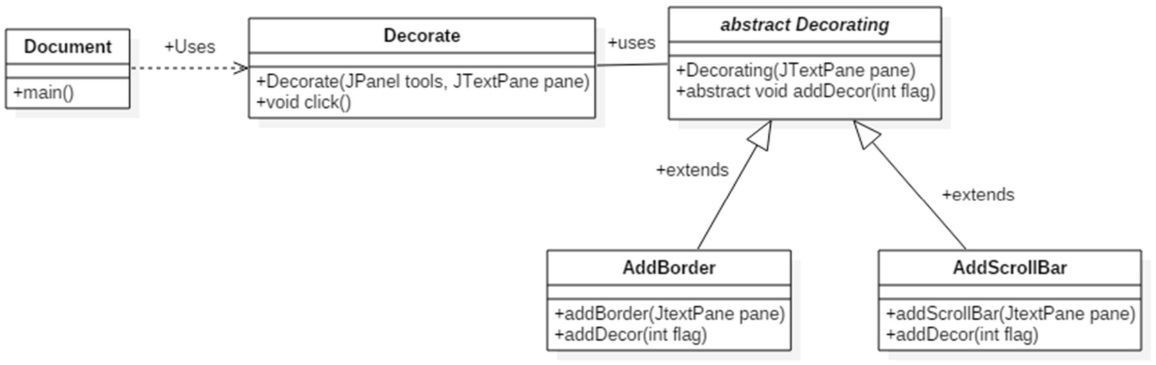
Participants

- Component
- ConcreteComponent
- Decorator[2][3]
- ConcreteDecorator[2][3]s

General Structure



Document Editor Specific Structure



Document
+main()
+Uses
Decorate
+Decorate(JPanel tools, JTextPane pane)
+void click()
+uses
abstract Decorating
+Decorating(JTextPane pane)
+abstract void addDecor(int flag)
+extends
+extends
AddBorder
+addBorder(JtextPane pane)
+addDecor(int flag)
AddScrollBar
+addScrollBar(JtextPane pane)
+addDecor(int flag)

Working in Code

Decorator[2][3] subclasses are free to add operations for specific functionality. For example, addScrollbar operation lets other objects scroll the interface if they know there happens to be a ScrollDecorator[2][3] object in the interface. The important aspect of this pattern is that it lets decorators appear anywhere a visual component can. That way, clients generally can't tell the difference between a decorated component and an undecorated one, and so they don't depend at all on the decoration.

Here, an abstract class Decorating is used to help add responsibilities dynamically such as AddBorder and AddScrollBar. Hence, depending on the instance of the decorating object the Decorator[2][3] class will provide the respective functionality to the Document (or Client) class.

3.4 Bridge Design Pattern

Intent

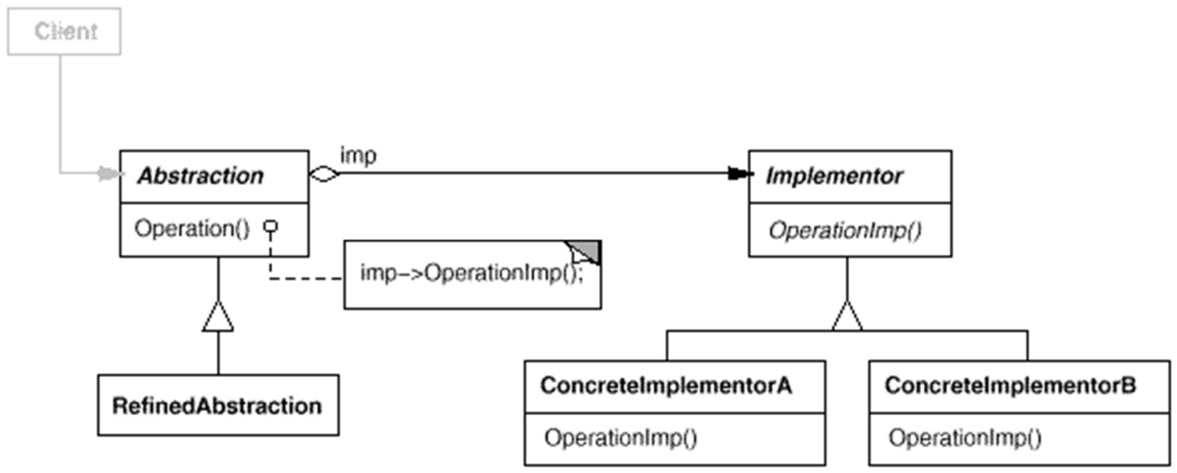
Decouple an abstraction from its implementation so that the two can vary independently. Publish interface in an inheritance hierarchy, and bury implementation in its own inheritance hierarchy. Beyond encapsulation, to insulation.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

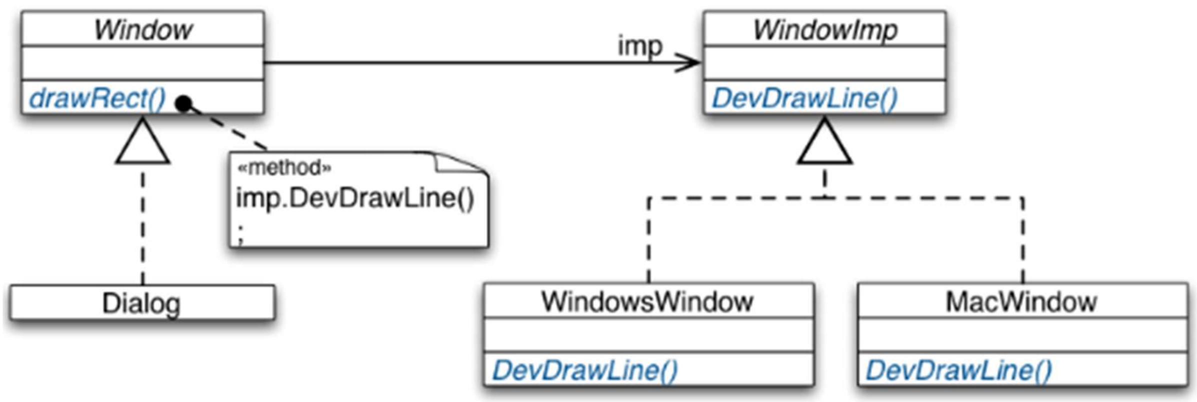
Motivation

When an abstraction can have one of several possible implementations, the usual way to accommodate them is to use inheritance. An abstract class defines the interface to the abstraction, and concrete subclasses implement it in different ways. But this approach isn't always flexible enough. Inheritance binds an implementation to the abstraction permanently, which makes it difficult to modify, extend, and reuse abstractions and implementations independently.

General Structure



Documents Editor Specific Structure



Working in Code

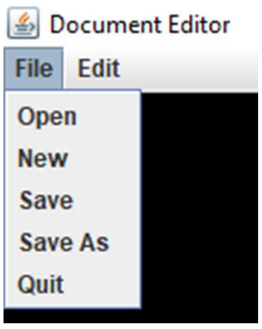
Bridge[4][5] design Pattern is used to decouple the abstraction from its implementation so that the two can vary independently. In the Document Editor application, the main aim of the Bridge[4][5] Design Pattern is to make the document that is generated to be made compatible with various operating systems. As the name suggests, the usage of Bridge[4][5] Design Pattern acts as a bridge between various platforms and fades the boundary of difference between implementation of various documents in different operating systems (Windows, Linux and MacOS respectively in this case).

4.0 Features

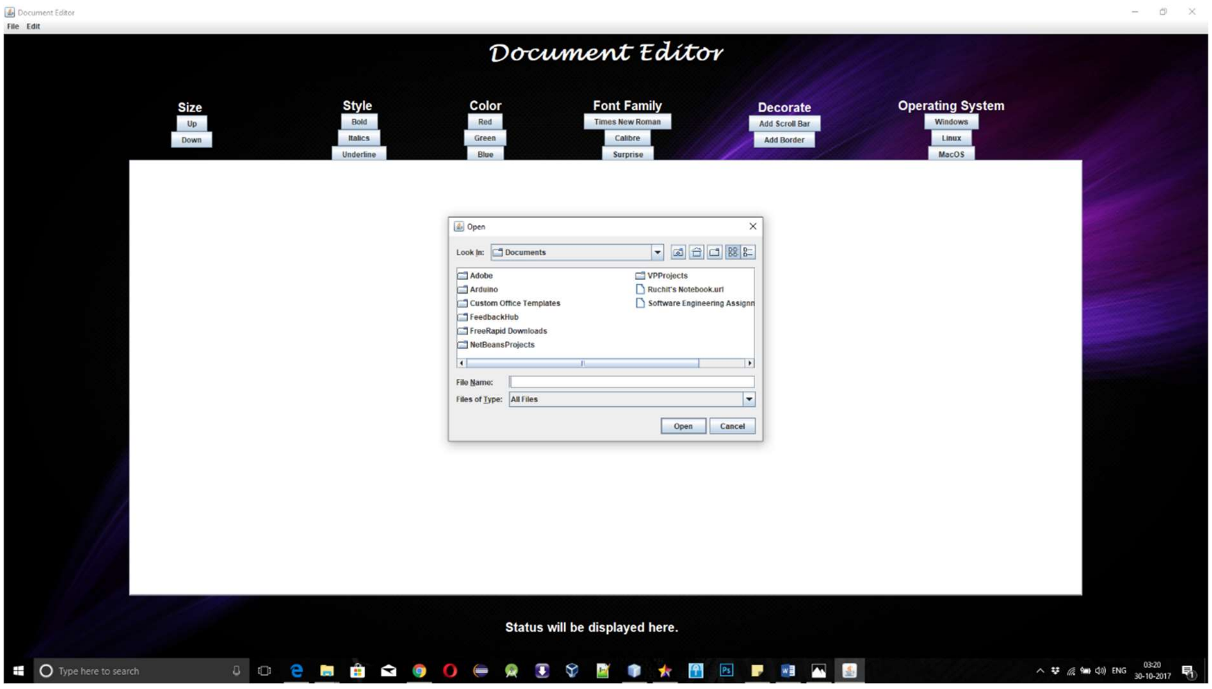
The salient features of the document editor are as follows –

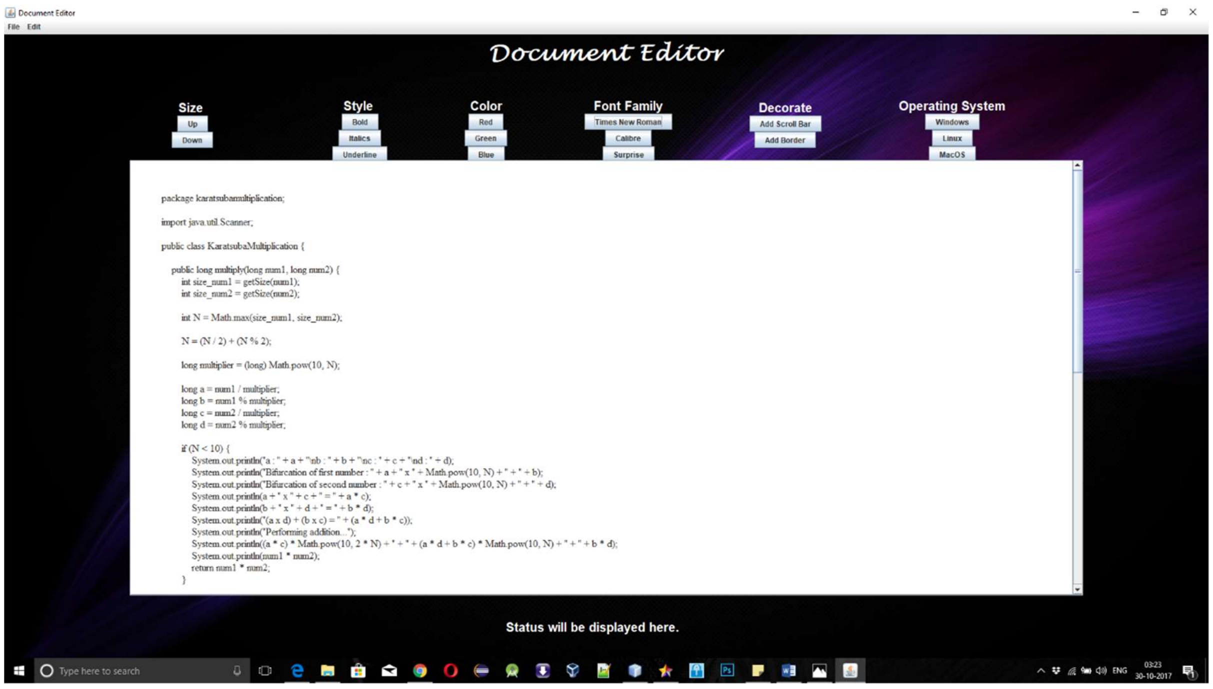
In the menu bar, the File menu comprises of Open, New, Save, Save As, Quit functionality whereas the Edit menu comprises of the Cut, Copy and Paste functionality.

File (Composite[8][9][9] Design Pattern)



- "Open" allows the document editor to open the documents present in that particular machine. Any kind of text document can be opened as long as it is a text file. Though the files having .docx extension are opened in an encrypted format.

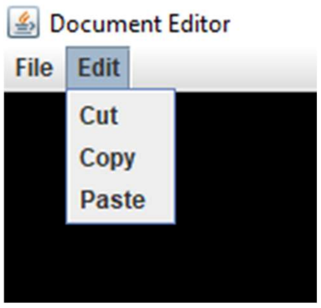




```
package karatsubamultiplication;

import java.util.Scanner;

public class KaratsubaMultiplication {

    public long multiply(long num1, long num2) {
        int size_num1 = getSize(num1);
        int size_num2 = getSize(num2);

        int N = Math.max(size_num1, size_num2);

        N = (N / 2) + (N % 2);

        long multiplier = (long) Math.pow(10, N);

        long a = num1 / multiplier;
        long b = num1 % multiplier;
        long c = num2 / multiplier;
        long d = num2 % multiplier;

        if (N < 10) {
            System.out.println("a : " + a + "\nb : " + b + "\nc : " + c + "\nd : " + d);
            System.out.println("Bifurcation of first number : " + a + " x " + Math.pow(10, N) + " + " + b);
            System.out.println("Bifurcation of second number : " + c + " x " + Math.pow(10, N) + " + " + d);
            System.out.println(a + " x " + c + " = " + a * c);
            System.out.println(b + " x " + d + " = " + b * d);
            System.out.println("(a x d) + (b x c) = " + (a * d + b * c));
            System.out.println("Performing addition...");
            System.out.println((a * c) * Math.pow(10, 2 * N) + " + " + (a * d + b * c) * Math.pow(10, N) + " + " + b * d);
            System.out.println(num1 * num2);
            return num1 * num2;
        }
```

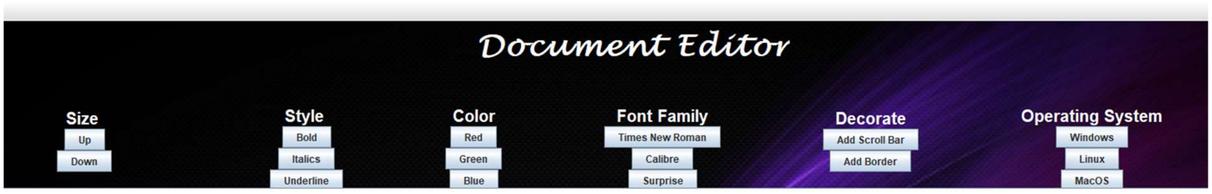
- “New” allows the user to make a new document. If the editor has something written to it, the document editor prompts the user to save the initial documents and then makes space for the new documents to be written.
- “Save” allows the documents to be saved.
- “Save As” allows the user to change the document type to that which is desired by the user. It again open the Save As dialog box.
- “Quit” allows the user to exit the Document Editor i.e. it disposes off the frame.

Edit (Composite[8][9][9] Design Pattern)



- “Cut” allows the selected text to be cut for further use and reference.
- “Copy” allows the selected text to be copied for further use and reference.
- “Paste” allows the text that was cut/copied to be pasted.

The “tools” panel consists of six set of buttons



<u>Size (Command[6][7] Design Pattern)</u>

- “Up” button increases the size of the text by font size 2 for every click of the button
- “Down” button decreases the size of the text by font size 2 for every click of the button.

<u>Style (Command[6][7] Design Pattern)</u>

- “Bold” bolds the text in the edit area of the document editor.
- “Italics” italicizes the text in the edit area of the document editor.
- “Underline” underlines the text in the edit area of the document editor.

<u>Color (Command[6][7] Design Pattern)</u>

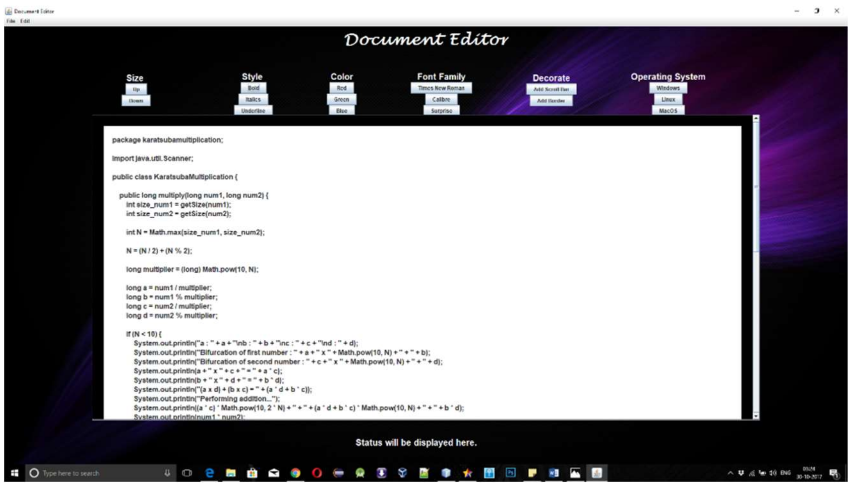
- “Red” button allows the text color of the text to be changed to the color red.
- “Green” button allows the text color of the text to be changed to the color Green.
- “Blue” button allows the text color of the text to be changed to the color Blue.

<u>Font Family (Command[6][7] Design Pattern)</u>

- “Times New Roman” changes the text style to that particular font family
- “Calibri” changes the text style to that particular font family
- “Surprise” changes the text style to OCR A Extended

<u>Decorator[2][3] (Decorator[2][3] Design Pattern</u>)

- “Add Border” adds a border to the edit area provided to the user to write in the text
- “Add Scroll Bar” adds a scroll bar when the text goes out of the current bounds of the edit area.

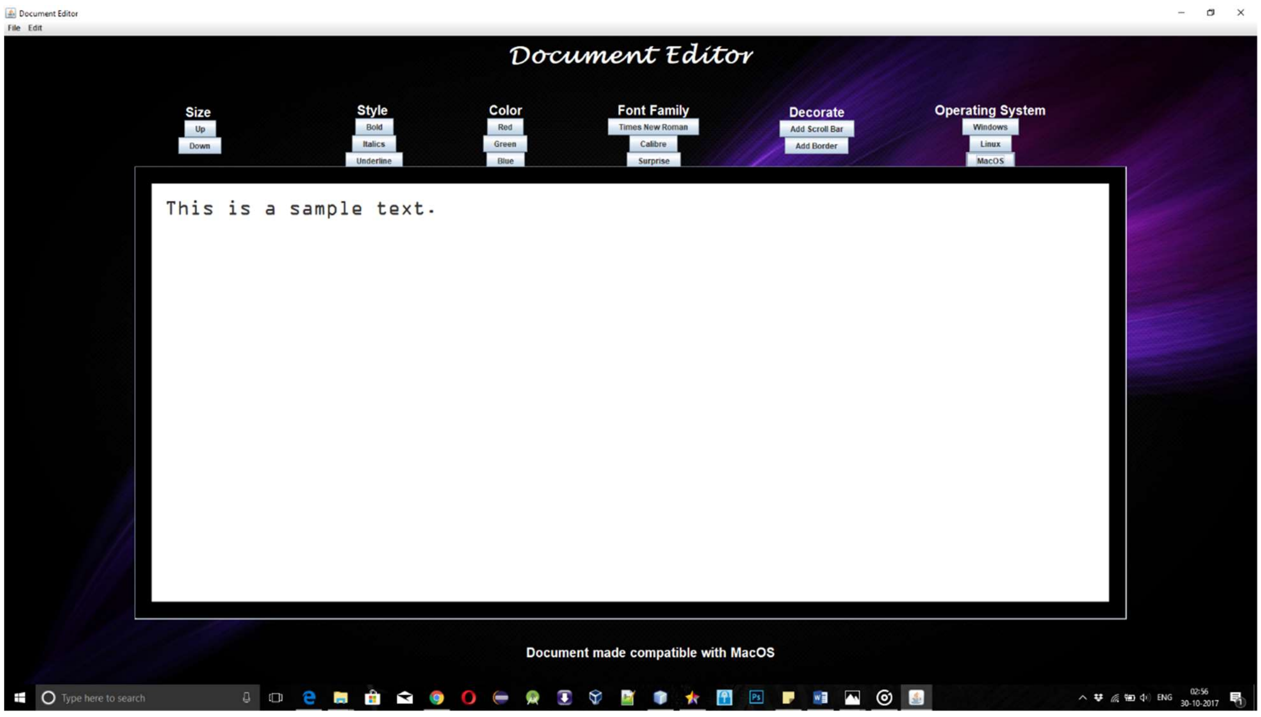


```
package karatsubamultiplication;

import java.util.Scanner;

public class KaratsubaMultiplication {

  public long multiply(long num1, long num2) {
    int size_num1 = getSize(num1);
    int size_num2 = getSize(num2);

    int N = Math.max(size_num1, size_num2);

    N = (N / 2) + (N % 2);

    long multiplier = (long) Math.pow(10, N);

    long a = num1 / multiplier;
    long b = num1 % multiplier;
    long c = num2 / multiplier;
    long d = num2 % multiplier;

    if (N < 10) {
      System.out.println("a : " + a + "\nb : " + b + "\nc : " + c + "\nd : " + d);
      System.out.println("Bifurcation of first number : " + a + " x " + Math.pow(10, N) + " + " + b);
      System.out.println("Bifurcation of second number : " + c + " x " + Math.pow(10, N) + " + " + d);
      System.out.println(a + " x " + c + " = " + a * c);
      System.out.println(b + " x " + d + " = " + b * d);
      System.out.println("(a x d) + (b x c) = " + (a * d + b * c));
      System.out.println("Performing addition...");
      System.out.println((a * c) * Math.pow(10, 2 * N) + " + " + (a * d + b * c) * Math.pow(10, N) + " + " + b * d);
      System.out.println(num1 * num2);
```

Status will be displayed here.

Operating System (Bridge[4][5] Design Pattern)

- "Windows" allows the document to be made compatible with Windows operating system. However in this particular implementation, only the label below the edit area shows the above scenario.
- "Linux" allows the document to be made compatible with Linux operating system. However in this particular implementation, only the label below the edit area shows the above scenario.
- "MacOS" allows the document to be made compatible with Mac operating system. However in this particular implementation, only the label below the edit area shows the above scenario.



```
This is a sample text.
```

Document made compatible with MacOS

5.0 Code

```
 package document;

//Imports are here
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultEditorKit;
import javax.swing.text.Style;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
import org.apache.poi.xwpf.extractor.XWPFWordExtractor;

//Composite[8][9][9] Design Pattern.
//The method is initialised and performed operation on in the PSVM
interface BaseComposite[8][9][9] {

    JPanel addComposite[8][9][9]();

    JTextPane addLeaf();
    Font f = new Font("Courier", Font.BOLD, 20);
    Font f1 = new Font("Lucida Handwriting", Font.BOLD, 36);
}

class Base implements BaseComposite[8][9][9] {

    JFrame frame;
    JPanel north, editPane, status, image, tools, menu, east, west, padding1, padding2, padding_bottom;
    JLabel message, background, title, foreground;
    JTextPane editarea2;
    int flagSave = 0;
    File fileSave = null;
    int clickCount = 0;

    Base() {
        frame = new JFrame("Document Editor");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        background = new JLabel(new
ImageIcon("C:\\Users\\ruch0\\OneDrive\\Documents\\NetBeansProjects\\Document\\src\\document\\bg7.jp
g"));
        frame.setContentPane(background);
        frame.setLayout(new BorderLayout());
        frame.pack();
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

    @Override
    public JPanel addComposite[8][9][9]() {
        north = new JPanel(new BorderLayout());
        editPane = new JPanel();
        status = new JPanel(new BorderLayout());
        image = new JPanel(new FlowLayout());
        tools = new JPanel(new BorderLayout());
        east = new JPanel();
        west = new JPanel();
        padding1 = new JPanel();
        padding2 = new JPanel();
        menu = new JPanel(new BorderLayout());
        message = new JLabel("Status will be displayed here.");
        message.setForeground(Color.WHITE);
        message.setFont(f);
        padding_bottom = new JPanel();
        title = new JLabel("Document Editor");
        title.setForeground(Color.WHITE);
        title.setFont(f1);

        frame.add(north, BorderLayout.NORTH);
        frame.add(editPane, BorderLayout.CENTER);
        frame.add(status, BorderLayout.SOUTH);
        frame.add(east, BorderLayout.EAST);
        frame.add(west, BorderLayout.WEST);
        north.add(image, BorderLayout.NORTH);
        north.add(menu, BorderLayout.SOUTH);
        image.add(title);
        status.add(padding_bottom, BorderLayout.WEST);
        status.add(message, BorderLayout.CENTER);
        menu.add(padding1, BorderLayout.WEST);
        menu.add(padding2, BorderLayout.EAST);
        menu.add(tools, BorderLayout.CENTER);
        tools.setLayout(new BoxLayout(tools, BoxLayout.X_AXIS));

        message.setPreferredSize(new Dimension(600, 200));
        padding_bottom.setPreferredSize(new Dimension(800, 200));
        image.setPreferredSize(new Dimension(500, 100));
        tools.setPreferredSize(new Dimension(500, 100));
        editPane.setPreferredSize(new Dimension(600, 150));
        status.setPreferredSize(new Dimension(100, 100));
        east.setPreferredSize(new Dimension(200, 100));
        west.setPreferredSize(new Dimension(200, 100));
        padding1.setPreferredSize(new Dimension(200, 100));
```

```
        padding2.setPreferredSize(new Dimension(200, 100));

        padding_bottom.setOpaque(false);
        menu.setOpaque(false);
        tools.setOpaque(false);
        east.setOpaque(false);
        west.setOpaque(false);
        status.setOpaque(false);
        padding1.setOpaque(false);
        padding2.setOpaque(false);
        image.setOpaque(false);
        editPane.setOpaque(false);
        north.setOpaque(false);
        return tools;
    }

    @Override
    public JTextPane addLeaf() {
        JMenuBar menubar = new JMenuBar();
        JMenu file = new JMenu("File");
        menubar.add(file);

        editarea2 = new JTextPane();
        editarea2.setMargin(new Insets(50, 50, 50, 50));
        editarea2.setPreferredSize(new Dimension(600, 600));
        JScrollPane scroll = new JScrollPane(editarea2);

        JMenuItem open = new JMenuItem("Open");
        file.add(open);
        open.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                XWPFWordExtractor extractor;
                JFileChooser fileChooser = new JFileChooser();
                int returnVal = fileChooser.showOpenDialog(frame);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File file = fileChooser.getSelectedFile();
                    try {
                        FileInputStream fis = new FileInputStream(file.getAbsolutePath());
                        BufferedReader input = new BufferedReader(new InputStreamReader(new
FileInputStream(file)));
                        editarea2.read(input, "READING FILE :-)");
                        fis.close();
                    } catch (Exception ex) {
                        System.out.println("problem accessing file" + file.getAbsolutePath());
                    }
                } else {
                    System.out.println("File access cancelled by user.");
                }
            }
        });

        JMenuItem neww = new JMenuItem("New");
```

```
file.add(neww);
neww.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if (editarea2.getText().isEmpty()) {
            editarea2.setText("");
        } else {
            save();
            editarea2.setText("");
            flagSave = 0;
            fileSave = null;
        }
    }
});

JMenuItem save = new JMenuItem("Save");
file.add(save);
save.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if (flagSave == 1) {
            BufferedWriter outFile = null;
            try {
                outFile = new BufferedWriter(new FileWriter(fileSave));
            } catch (IOException ex) {
                Logger.getLogger(Document.class.getName()).log(Level.SEVERE, null, ex);
            }
            try {
                editarea2.write(outFile);
            } catch (IOException ex) {
                Logger.getLogger(Document.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                if (outFile != null) {
                    try {
                        outFile.close();
                    } catch (IOException e) {
                    }
                }
            }
        } else {
            save();
        }
    }
});

JMenuItem saveas = new JMenuItem("Save As");
file.add(saveas);
saveas.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        save();
    }
});
```

```
    JMenuItem quit = new JMenuItem("Quit");
    file.add(quit);
    quit.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            frame.dispose();
        }
    });

    JMenu edit = new JMenu("Edit");
    menubar.add(edit);
    JMenuItem undo = new JMenuItem("Undo");
    edit.add(undo);
    undo.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {

        }
    });

    JMenuItem cut = new JMenuItem(new DefaultEditorKit.CutAction());
    cut.setText("Cut");
    edit.add(cut);

    JMenuItem copy = new JMenuItem(new DefaultEditorKit.CopyAction());
    copy.setText("Copy");
    edit.add(copy);

    JMenuItem paste = new JMenuItem(new DefaultEditorKit.PasteAction());
    paste.setText("Paste");
    edit.add(paste);

    JRootPane root = image.getRootPane();
    root.setJMenuBar(menubar);
    JRootPane root2 = editPane.getRootPane();

    root2.getContentPane().add(scroll);
    editPane.setVisible(true);
    return editarea2;
}

//Method to perform the save as operation on the text
public void save() {
    FileNameExtensionFilter extensionFilter = new FileNameExtensionFilter("Text File", "txt");
    final JFileChooser saveAsFileChooser = new JFileChooser();
    saveAsFileChooser.setApproveButtonText("Save");
    saveAsFileChooser.setFileFilter(extensionFilter);
    int actionDialog = saveAsFileChooser.showOpenDialog(frame);
    if (actionDialog != JFileChooser.APPROVE_OPTION) {
        return;
    }
```

```
        File file = saveAsFileChooser.getSelectedFile();
        if (!file.getName().endsWith(".txt")) {
            file = new File(file.getAbsolutePath() + ".txt");
        }

        BufferedWriter outFile = null;
        try {
            outFile = new BufferedWriter(new FileWriter(file));
        } catch (IOException ex) {
            System.out.println("Exception Occurred");
        }

        try {
            editarea2.write(outFile);
        } catch (IOException ex) {
            System.out.println("Exception occurred");
        } finally {
            if (outFile != null) {
                try {
                    outFile.close();
                } catch (IOException e) {
                }
            }

        }
    }
}

//Command Design Pattern.
//The invoker class in written below and it is called in the PSVM
interface ButtonsCommand {

    void click();
    Font f = new Font("Courier", Font.BOLD, 20);
    Dimension dimen = new Dimension(200, 200);

}

class FontSize implements ButtonsCommand {

    JButton incr, decr;
    JPanel tools, p1;
    JTextPane pane;
    JLabel l1, label;
    static int a;

    FontSize(JPanel panel, JTextPane pane) {
        p1 = new JPanel();
        p1.setPreferredSize(dimen);
        p1.setOpaque(false);
        this.tools = panel;
        this.pane = pane;
        incr = new JButton("Up");
        incr.setAlignmentX(Component.CENTER_ALIGNMENT);
        decr = new JButton("Down");
        decr.setAlignmentX(Component.CENTER_ALIGNMENT);
        l1 = new JLabel("Size ");
        l1.setAlignmentX(Component.CENTER_ALIGNMENT);
        l1.setForeground(Color.WHITE);
        label = new JLabel("\t\t\t\t");

        StyledDocument doc = (StyledDocument) pane.getDocument();
        Style style = doc.addStyle("StyleName", null);
        a = StyleConstants.getFontSize(style);
    }

    @Override
    public void click() {

        p1.setLayout(new BoxLayout(p1, BoxLayout.Y_AXIS));
        l1.setFont(f);
        p1.add(l1);
        p1.add(incr, Component.CENTER_ALIGNMENT);
        p1.add(decr, Component.CENTER_ALIGNMENT);
        p1.add(label);
        JPanel temp = new JPanel();
        temp.setOpaque(false);
        temp.setSize(new Dimension(50, 50));
        tools.add(temp, BorderLayout.WEST);
        temp.setBackground(Color.ORANGE);
        tools.add(p1);
        temp.setSize(new Dimension(50, 50));
        tools.add(temp);
        p1.setBackground(Color.ORANGE);

        incr.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                Font font = pane.getFont();
                float size = font.getSize() + 2.0f;
                pane.setFont(font.deriveFont(size));
            }
        });

        decr.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                Font font = pane.getFont();
                float size = font.getSize() - 2.0f;
                pane.setFont(font.deriveFont(size));
            }
        });
    }
}
```

```
class FontStyle implements ButtonsCommand {

    JButton bold, italics, underline;
    JPanel tools, p2;
    JTextPane pane;
    JLabel l2;

    FontStyle(JPanel panel, JTextPane pane) {
        p2 = new JPanel();
        p2.setOpaque(false);
        this.tools = panel;
        this.pane = pane;
        bold = new JButton("Bold");
        bold.setAlignmentX(Component.CENTER_ALIGNMENT);
        italics = new JButton("Italics");
        italics.setAlignmentX(Component.CENTER_ALIGNMENT);
        underline = new JButton("Underline");
        underline.setAlignmentX(Component.CENTER_ALIGNMENT);
        l2 = new JLabel("Style ");
        l2.setForeground(Color.WHITE);
        l2.setAlignmentX(Component.CENTER_ALIGNMENT);
    }

    @Override
    public void click() {
        p2.setLayout(new BoxLayout(p2, BoxLayout.Y_AXIS));
        l2.setFont(f);
        p2.add(l2);
        p2.add(bold);
        p2.add(italics);
        p2.add(underline);
        tools.add(p2);
        JPanel temp = new JPanel();
        temp.setOpaque(false);
        temp.setSize(new Dimension(50, 50));
        temp.setBackground(Color.ORANGE);
        tools.add(temp);
        p2.setBackground(Color.ORANGE);

        bold.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setBold(style, true);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        });
        italics.addActionListener(
                new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae
            ) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setItalic(style, true);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        }
        );

        underline.addActionListener(
                new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae
            ) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setUnderline(style, true);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        }
        );
    }

}

class FontColor implements ButtonsCommand {

    JPanel tools, p3;
    JTextPane pane;
    JButton red, green, blue;
    JLabel l3;

    FontColor(JPanel tools, JTextPane pane) {
        this.pane = pane;
        this.tools = tools;
        p3 = new JPanel();
        p3.setOpaque(false);
        red = new JButton("Red");
        red.setAlignmentX(Component.CENTER_ALIGNMENT);
        green = new JButton("Green");
        green.setAlignmentX(Component.CENTER_ALIGNMENT);
        blue = new JButton("Blue");
        blue.setAlignmentX(Component.CENTER_ALIGNMENT);
        l3 = new JLabel("Color");
        l3.setForeground(Color.WHITE);
        l3.setAlignmentX(Component.CENTER_ALIGNMENT);
    }

    @Override
    public void click() {
        p3.setLayout(new BoxLayout(p3, BoxLayout.Y_AXIS));
        l3.setFont(f);
        p3.add(l3);
        p3.add(red);
        p3.add(green);
        p3.add(blue);
        tools.add(p3);
        JPanel temp = new JPanel();
        temp.setOpaque(false);
        temp.setSize(new Dimension(50, 50));
        tools.add(temp);
        temp.setBackground(Color.ORANGE);
        p3.setBackground(Color.ORANGE);

        red.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setForeground(style, Color.RED);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        });

        green.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setForeground(style, Color.GREEN);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        });

        blue.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                StyledDocument doc = (StyledDocument) pane.getDocument();
                Style style = doc.addStyle("StyleName", null);
                StyleConstants.setForeground(style, Color.BLUE);
                try {
                    String text = pane.getDocument().getText(0, pane.getDocument().getLength());
                    pane.setText("");
                    doc.insertString(0, text, style);
                } catch (BadLocationException ex) {
                }
            }
        });

    }
}

class FontFamily implements ButtonsCommand {

    JPanel tools, p4;
    JTextPane pane;
    JButton tnr, calibre, sansSerif;
    JLabel l4;

    FontFamily(JPanel tools, JTextPane pane) {
        this.pane = pane;
        this.tools = tools;
        p4 = new JPanel();
        p4.setOpaque(false);
        tnr = new JButton("Times New Roman");
        tnr.setAlignmentX(Component.CENTER_ALIGNMENT);
        calibre = new JButton("Calibre");
        calibre.setAlignmentX(Component.CENTER_ALIGNMENT);
        sansSerif = new JButton("Surprise");
        sansSerif.setAlignmentX(Component.CENTER_ALIGNMENT);
        l4 = new JLabel("Font Family");
        l4.setForeground(Color.WHITE);
        l4.setAlignmentX(Component.CENTER_ALIGNMENT);
    }

    @Override
    public void click() {
        p4.setLayout(new BoxLayout(p4, BoxLayout.Y_AXIS));
        l4.setFont(f);
        p4.add(l4);
        p4.add(tnr);
        p4.add(calibre);
        p4.add(sansSerif);
        tools.add(p4);
    JPanel temp = new JPanel();
    temp.setOpaque(false);
    temp.setSize(new Dimension(50, 50));
    tools.add(temp);
    temp.setBackground(Color.ORANGE);
    p4.setBackground(Color.ORANGE);

    tnr.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent ae) {
        StyledDocument doc = (StyledDocument) pane.getDocument();
        Style style = doc.addStyle("StyleName", null);
        StyleConstants.setFontFamily(style, "Times New Roman");
        try {
          String text = pane.getDocument().getText(0, pane.getDocument().getLength());
          pane.setText("");
          doc.insertString(0, text, style);
        } catch (BadLocationException ex) {
        }
      }
    });

    calibre.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent ae) {
        StyledDocument doc = (StyledDocument) pane.getDocument();
        Style style = doc.addStyle("StyleName", null);
        StyleConstants.setFontFamily(style, "Calibri");
        try {
          String text = pane.getDocument().getText(0, pane.getDocument().getLength());
          pane.setText("");
          doc.insertString(0, text, style);
        } catch (BadLocationException ex) {
        }
      }
    });

    sansSerif.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent ae) {
        StyledDocument doc = (StyledDocument) pane.getDocument();
        Style style = doc.addStyle("StyleName", null);
        StyleConstants.setFontSize(style, 32);
        StyleConstants.setFontFamily(style, "OCR A Extended");
        try {
          String text = pane.getDocument().getText(0, pane.getDocument().getLength());
          pane.setText("");
          doc.insertString(0, text, style);
        } catch (BadLocationException ex) {
        }
      }
    });
  }
```

```
}

//This implementation includes the use of the Decorator[2][3][2][3] Design Pattern as well.
class Decorate implements ButtonsCommand {

  JPanel tools, p5;
  JTextPane pane;
  JButton addscbar, addBorder;
  JLabel l5, label;

  Decorate(JPanel tools, JTextPane pane) {
    this.pane = pane;
    this.tools = tools;
    p5 = new JPanel();
    label = new JLabel("\t\t\t\t");
    p5.setOpaque(false);
    addscbar = new JButton("Add Scroll Bar");
    addscbar.setAlignmentX(Component.CENTER_ALIGNMENT);
    addBorder = new JButton("Add Border");
    addBorder.setAlignmentX(Component.CENTER_ALIGNMENT);
    l5 = new JLabel("Decorate");
    l5.setForeground(Color.WHITE);
    l5.setAlignmentX(Component.CENTER_ALIGNMENT);
  }

  @Override
  public void click() {
    p5.setLayout(new BoxLayout(p5, BoxLayout.Y_AXIS));
    l5.setFont(f);
    p5.add(l5);
    p5.add(addscbar);
    p5.add(addBorder);
    p5.add(label);
    tools.add(p5);
    JPanel temp = new JPanel();
    temp.setOpaque(false);
    temp.setSize(new Dimension(50, 50));
    tools.add(temp);
    temp.setBackground(Color.ORANGE);
    p5.setBackground(Color.ORANGE);

    addBorder.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent ae) {
        //Instantiating the Deorating class in the Decorator[2][3][2][3] Design Pattern
        Decorating d = new AddBorder(pane);
        d.addDecor(0);
      }
    });

    addscbar.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent ae) {
```

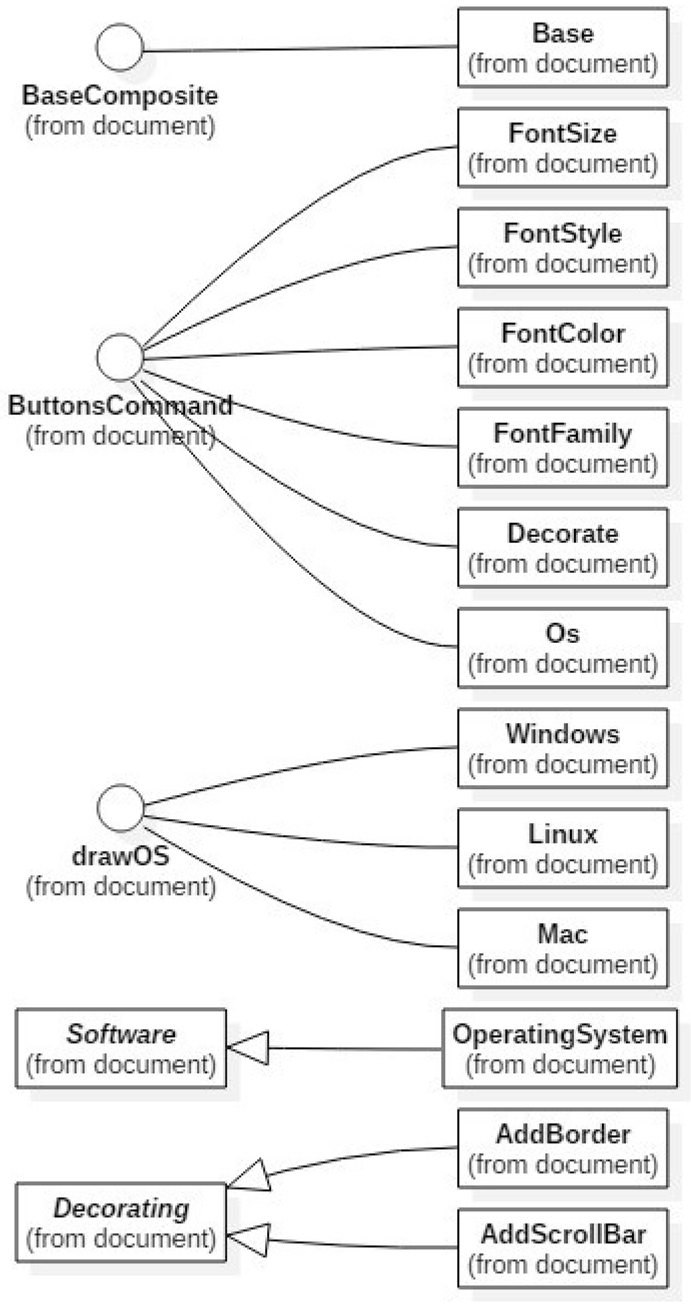
```
                //Put code for adding scroll bar
            }
        });
    }
}

//This implementation includes the use of the Bridge[4][5] Design Pattern as well.
class Os implements ButtonsCommand {

    JPanel tools, p6;
    JTextPane pane;
    JButton windows, linux, macOS;
    JLabel status, l2;

    Os(JPanel tools, JTextPane pane, JLabel status) {
        this.pane = pane;
        this.tools = tools;
        this.status = status;
        p6 = new JPanel();
        p6.setOpaque(false);
        windows = new JButton("Windows");
        windows.setAlignmentX(Component.CENTER_ALIGNMENT);
        linux = new JButton("Linux");
        linux.setAlignmentX(Component.CENTER_ALIGNMENT);
        macOS = new JButton("MacOS");
        macOS.setAlignmentX(Component.CENTER_ALIGNMENT);
        l2 = new JLabel("Operating System");
        l2.setForeground(Color.WHITE);
        l2.setAlignmentX(Component.CENTER_ALIGNMENT);
    }

    @Override
    public void click() {
        p6.setLayout(new BoxLayout(p6, BoxLayout.Y_AXIS));
        l2.setFont(f);
        p6.add(l2);
        p6.add(windows);
        p6.add(linux);
        p6.add(macOS);
        tools.add(p6);
        JPanel temp = new JPanel();
        temp.setOpaque(false);
        temp.setSize(new Dimension(50, 50));
        tools.add(temp);
        temp.setBackground(Color.ORANGE);
        p6.setBackground(Color.ORANGE);

        windows.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                //Instantiating the abstract Software class and passing the information of the type of OS and Platform
in the parameters
                Software windows = new OperatingSystem("Windows", "Platform1", new Windows(), status);
                windows.display();
            }
        });

        linux.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                //Instantiating the abstract Software class and passing the information of the type of OS and Platform
in the parameters
                Software linux = new OperatingSystem("Linux", "Platform2", new Linux(), status);
                linux.display();
            }
        });

        macOS.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                //Instantiating the abstract Software class and passing the information of the type of OS and Platform
in the parameters
                Software macOS = new OperatingSystem("MacOS", "Platform3", new Mac(), status);
                macOS.display();
            }
        });
    }
}

//Invoker class for the Command Design Pattern which has been used to add buttons
class Invoke {

    private java.util.List<ButtonsCommand> orderList = new ArrayList<>();

    public void addButton(ButtonsCommand button) {
        orderList.add(button);
    }

    public void placeButton() {
        for (ButtonsCommand buttonsCommand : orderList) {
            buttonsCommand.click();
        }
    }
}

//Bridge[4][5] Design Pattern
interface drawOS {

    public void displayOS(String os, String platform, JLabel message);
}

class Windows implements drawOS {

    @Override
    public void displayOS(String os, String platform, JLabel status) {
        status.setText("Document made compatible with Windows OS");
```

```
    }

}

class Linux implements drawOS {

    @Override
    public void displayOS(String os, String platform, JLabel message) {
        message.setText("Document made compatible with LINUX OS");
    }

}

class Mac implements drawOS {

    @Override
    public void displayOS(String os, String platform, JLabel message) {
        message.setText("Document made compatible with MacOS");
    }

}

abstract class Software {

    private drawOS dos;

    Software(drawOS dos) {
        this.dos = dos;
    }

    public abstract void display();
}

class OperatingSystem extends Software {

    String os, platform;
    drawOS dos;
    JLabel message;

    public OperatingSystem(String os, String playform, drawOS dos, JLabel message) {
        super(dos);
        this.dos = dos;
        this.os = os;
        this.platform = platform;
        this.message = message;
    }

    @Override
    public void display() {
        dos.displayOS(os, platform, message);
    }
}
```

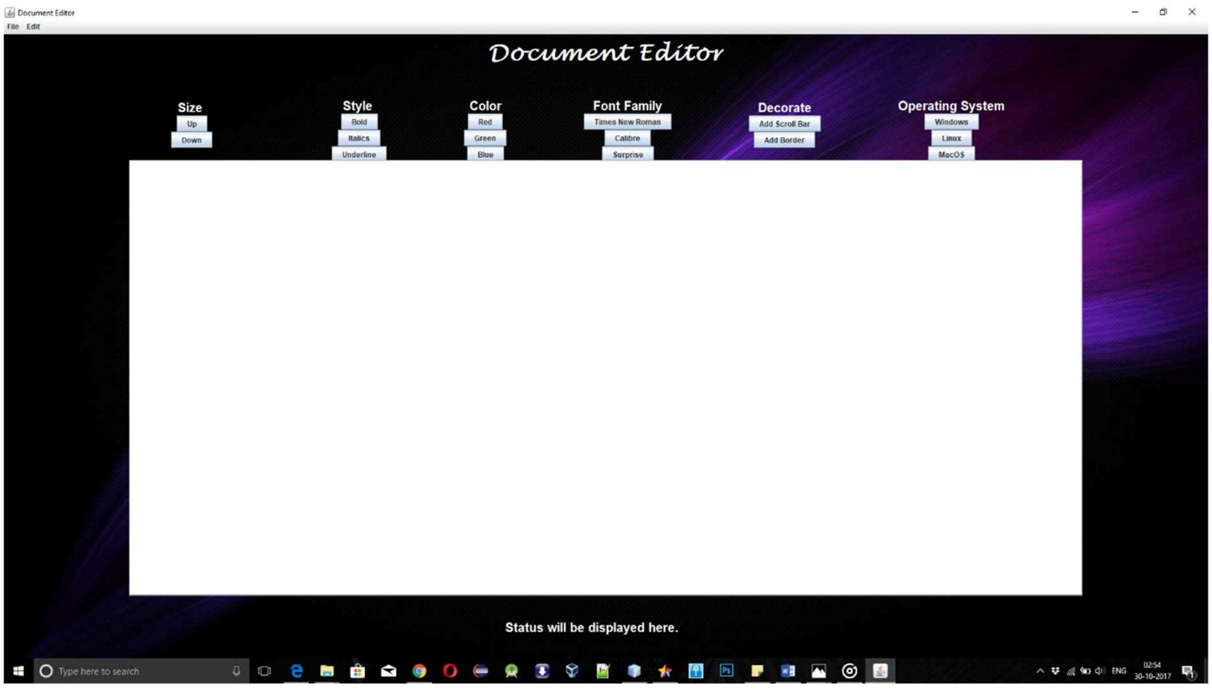
```
//Decorator[2][3][2][3] Design Pattern
abstract class Decorating {

   JTextPane pane;

   Decorating(JTextPane pane) {
      this.pane = pane;
   }

   abstract void addDecor(int flag);
}

class AddBorder extends Decorating {

   public AddBorder(JTextPane pane) {
      super(pane);
      this.pane = pane;
   }

   @Override
   void addDecor(int flag) {
      if (flag % 2 == 0) {
         pane.setMargin(new Insets(100, 100, 100, 100));
         pane.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.BLACK, 
25), BorderFactory.createEmptyBorder(20, 20, 20, 20)));
      } else {
         pane.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.white), 
BorderFactory.createEmptyBorder(10, 10, 10, 10)));
      }
   }
}

class AddScrollBar extends Decorating {

   public AddScrollBar(JTextPane pane) {
      super(pane);
      this.pane = pane;
   }

   @Override
   void addDecor(int flag) {
      throw new UnsupportedOperationException("Not supported yet."); //To change body of generated 
methods, choose Tools | Templates.
   }

}

public class Document {

   public static void main(String[] args) {
      //Instantiating the Base class in the Composite[8][9][9] Design Pattern
      Base b = new Base();
```

```
        //Collecting objects of JPanel & JTextPane
        JPanel p = b.addComposite[8][9][9]();
        JTextPane t = b.addLeaf();

        //instantiating all classes used to add functionalities using Command Design Pattern
        FontSize fs = new FontSize(p, t);
        FontStyle st = new FontStyle(p, t);
        FontColor clr = new FontColor(p, t);
        FontFamily fam = new FontFamily(p, t);
        Decorate dr = new Decorate(p, t);
        Os os = new Os(p, t, b.message);

        //Instantiating the Invoker class and adding all buttons to the array list
        Invoke iv = new Invoke();
        iv.addButton(fs);
        iv.addButton(st);
        iv.addButton(clr);
        iv.addButton(fam);
        iv.addButton(dr);
        iv.addButton(os);

        //Placing the buttons on the JPanel using placeButton() method
        iv.placeButton();
    }
}
```

6.0 Overall Class Diagram (Type Hierarchy)



7.0 Output Screenshot



Status will be displayed here.

8.0 References

[1] Erich Gamma, et al., Design Patterns[1]: Elements of Reusable Object-Oriented Software, U.S.A.: Addison-Wesley, August 1994.

[2] Sourcemaking.com, ‘Design Patterns[1] and Refactoring’. [Online]. Available: https://sourcemaking.com/design_patterns/decorator. [Accessed: 20- Oct- 2017].

[3] www.tutorialspoint.com, ‘Design Patterns[1] Composite[8][9][9] Pattern’. [Online]. Available: https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm. [Accessed: 20- Oct- 2017].

[4] Sourcemaking.com, ‘Design Patterns[1] and Refactoring’. [Online]. Available: https://sourcemaking.com/design_patterns/bridge. [Accessed: 22- Oct- 2017].

[5] www.tutorialspoint.com, ‘Design Patterns[1] Composite[8][9][9] Pattern’. [Online]. Available: https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm. [Accessed: 22- Oct- 2017].

[6] Sourcemaking.com, ‘Design Patterns[1] and Refactoring’. [Online]. Available: https://sourcemaking.com/design_patterns/command. [Accessed: 23- Oct- 2017].

[7] www.tutorialspoint.com, ‘Design Patterns[1] Composite[8][9][9] Pattern’. [Online]. Available: https://www.tutorialspoint.com/design_pattern/command_pattern.htm. [Accessed: 23- Oct- 2017].

[8] Sourcemaking.com, ‘Design Patterns[1] and Refactoring’. [Online]. Available: https://sourcemaking.com/design_patterns/composite. [Accessed: 25- Oct- 2017].

[9] www.tutorialspoint.com, ‘Design Patterns[1] Composite[8][9][9] Pattern’. [Online]. Available: https://www.tutorialspoint.com/design_pattern/composite_pattern.htm. [Accessed: 25- Oct- 2017].