Object-Oriented Software Engineering

Instructor : Huang, Chuen-Min

Teamwork2 ver.1

Group 2

ID	Name
B10821124	Leo
B10823011	Kousa
B10823015	Debby
B10823016	Kris
B10823018	Bob
B10823024	Michael
B10823029	Leo
B10823031	Andrew
B10823038	Joanne
B11023069	Young

Date 2021/12/29

Snapshots of the new result

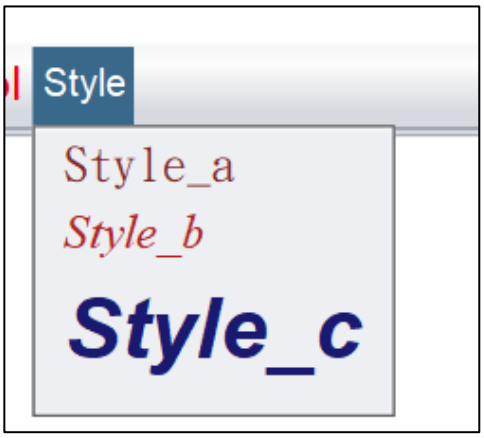


Figure 1. Use Template method pattern to add three new font styles we can apply.

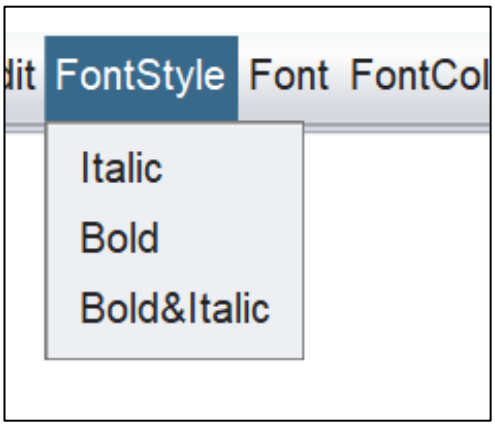


Figure 2. Use Decorator pattern to let Italic and Bold can use separately.

Figure 3. Use Observer pattern to count of line and display at the bottom state label.

Snapshots of new result

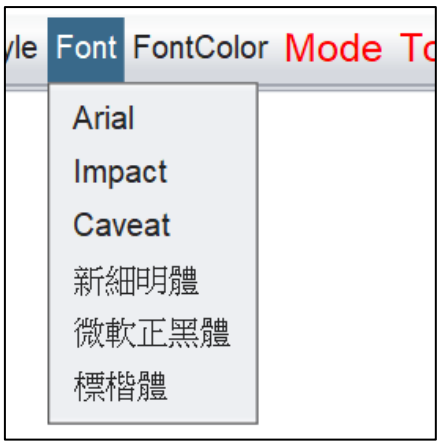
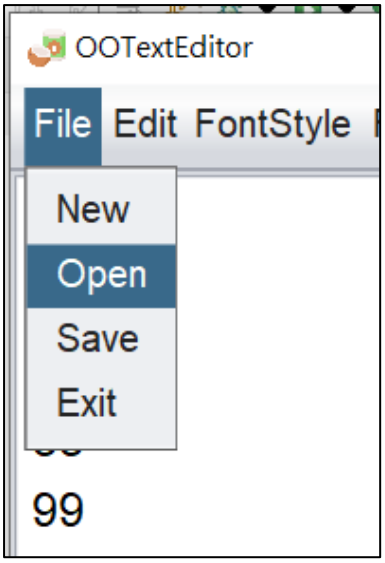


Figure 4. Add three new fonts can apply.



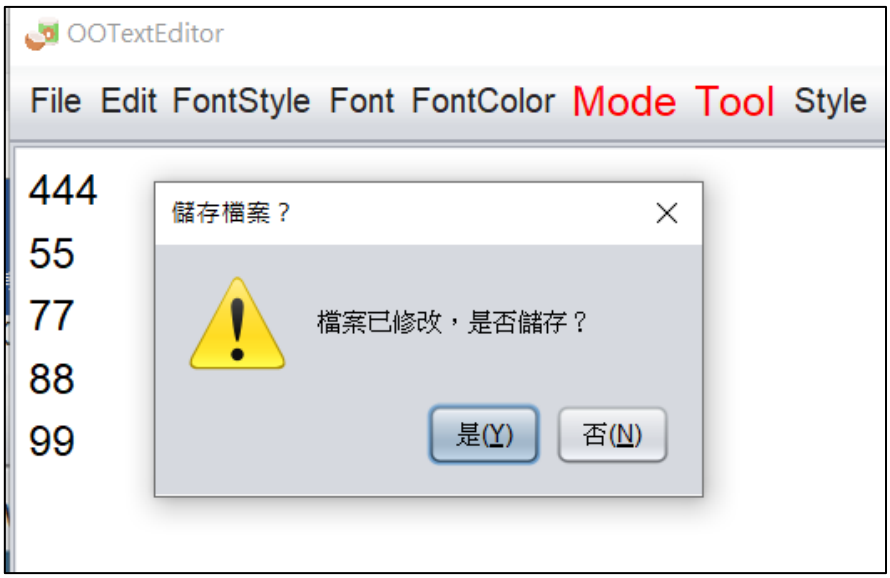


Figure 5&6 If we click open or exit while the file is not saved, it will show the dialog.

New pattern

Template Method

We use Template pattern to change font, font style, size, and color.

In class “StyleTemplate”, method “actionPerformed()” is a final method. It defines the order of executing the methods, subclasses can’t be changed, they must follow this process.

Method “changesize()” is Hook Method. It can be overridden by subclasses. If changesize() is true, we will do setfontsize().

```
abstract public class StyleTemplate implements ActionListener{
    JTextArea textArea1;
    Font f;
    public StyleTemplate(JTextArea t){
        textArea1 = t; //get JTextArea
    }
    public void setcolor() {
        textArea1.setForeground(new Color(0,0,0));
    }
    public void setfontname() {
        f = textArea1.getFont();
        textArea1.setFont(new Font("Arial",f.getStyle(),f.getSize()));
    }
    public void setfontstyle() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),Font.PLAIN,f.getSize()));
    }
    public void setfontsize() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),f.getStyle(),20));
    }
    public boolean changesize() {   //hookMethod
        return false;
    }
    final public void actionPerformed(ActionEvent e) { //TemplateMethod
        setcolor();
        setfontname();
        setfontstyle();
        if(changesize()) {
        setfontsize();
        }
     }
}
```

New pattern – Template Method

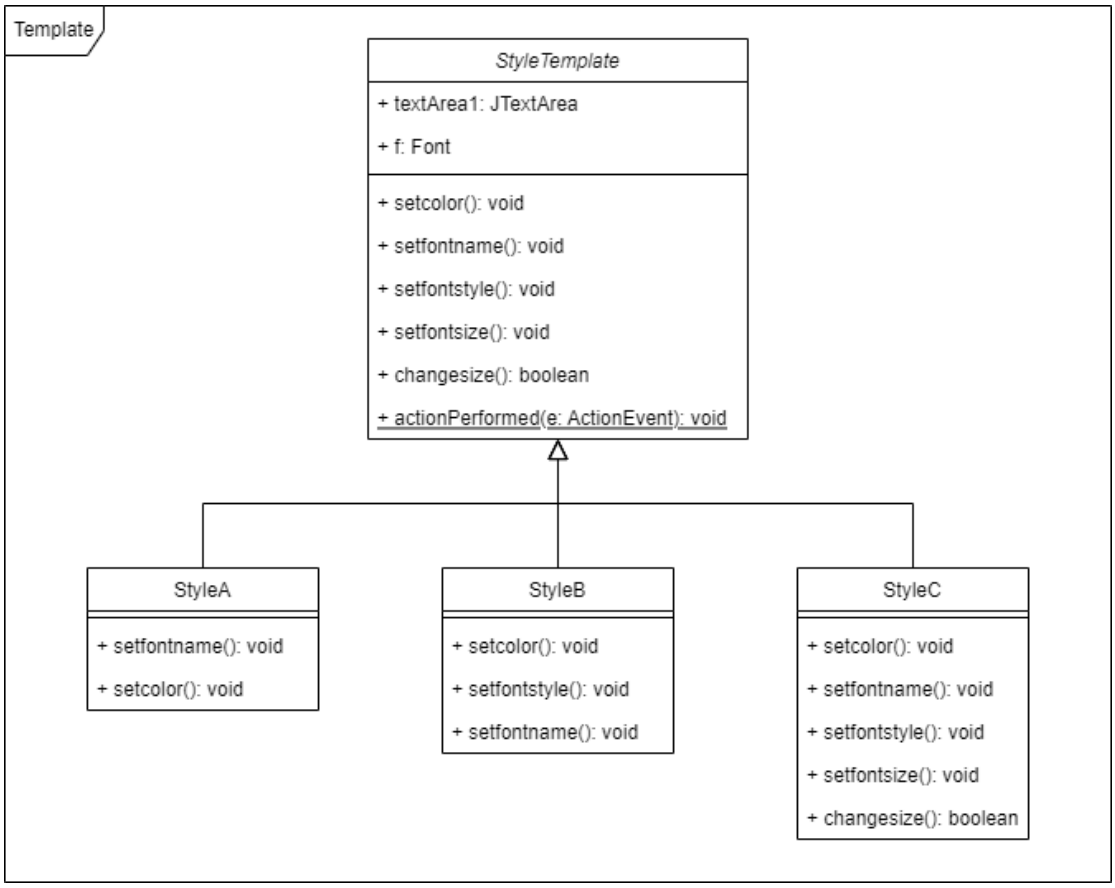
For example, we want to change font size in class “StyleC”, so we override changesize() to let it be true, then we can use setfontsize() to change the font size.

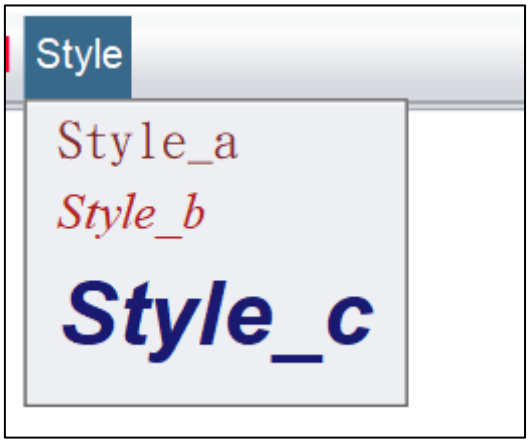
```
public class StyleC extends StyleTemplate{
    public StyleC(JTextArea t){
        super(t);
    }
    @Override
    public void setcolor() {
        textArea1.setForeground(new Color(25,25,112));
    }
    @Override
    public void setfontname() {
        f = textArea1.getFont();
        textArea1.setFont(new Font("宋體",f.getStyle(),f.getSize()));
    }
    @Override
    public void setfontstyle() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),Font.BOLD+Font.ITALIC,f.getSize()));
    }
    @Override
    public void setfontsize() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),f.getStyle(),36));
    }
    @Override
    public boolean changesize() {
        return true;
    }
}
```

And we don’t want to change font size in class “StyleA”, so we won’t override changesize().

```
public class StyleA extends StyleTemplate{
    public StyleA(JTextArea t){
        super(t);
    }
    @Override
    public void setfontname() {
        f = textArea1.getFont();
        textArea1.setFont(new Font("標楷體",f.getStyle(),f.getSize()));
    }
    @Override
    public void setcolor() {
        textArea1.setForeground(new Color(128,42,42));
    }
}
```

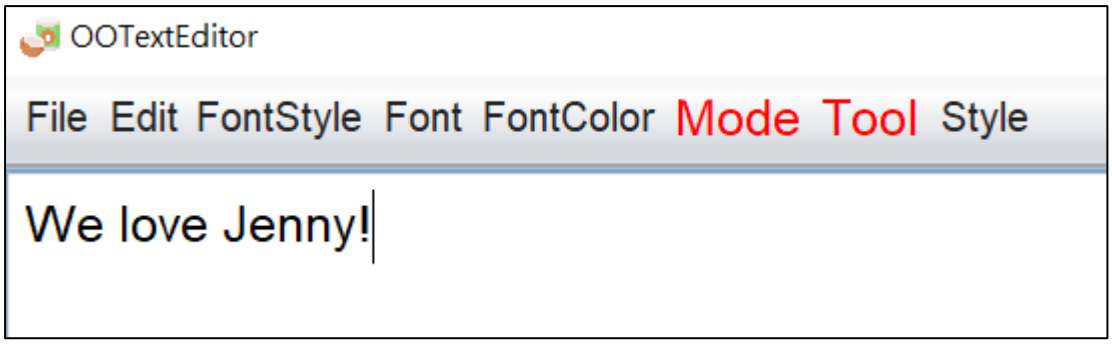
New pattern – Template Method





We have three styles to apply.

New pattern – Template Method



Text before applying style.

StyleA

StyleB

StyleC

New pattern

Chain of Responsibility

Chain of Responsibility pattern avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

In our system, the abstract class “fileHandler” has an abstract method for processing requests and a method for connecting with successors.

In method tonext(), if we don't have next successors, we will close the dialog.

```
//handler要實作的介面
public abstract class fileHandler {
    //每個人都要知道下一個要處理的人是誰
    public fileHandler next=null;

    fileHandler(fileHandler next){
        this.next=next;
    }
    //傳入目前存檔的狀態用來判斷
    //傳入textarea跟GUIfacade是儲存跟開啟檔案的語法需要
    public abstract void run(String state,JTextArea t,GUIfacade j);
    //傳給下一個人
    public void tonext(String state,JTextArea t,GUIfacade j) {
        if(next!=null) {
            next.run(state,t,j);
        }else {
            j.closeDialog();
        }
    }
}
```

The class “openHandler”, “exitHandler”, “saveHandler” and “unsaveHandler” extend the abstract class “fileHandler”, and these classes are responsible for handling requests from client.

New pattern- Chain of Responsibility

In class “unsaveHandler”, the method run() will use state to judge whether the file is saved. If the state is “no”, which means the file hasn't been saved yet, then we will ask whether the user wants to save it or not, and pass the state to next successor.

```
public class unsaveHandler extends fileHandler {

    public unsaveHandler(fileHandler next) {
        super(next);
    }
    public void run(String state,JTextArea t,GUIfacade j) {
        if(state=="no") {
            int option =JOptionPane.showConfirmDialog(
            null,"檔案已修改，是否儲存？",
            "儲存檔案？",JOptionPane.YES_NO_OPTION,
            JOptionPane.WARNING_MESSAGE,null);

            switch(option){
            // 確認檔案儲存
            case JOptionPane.YES_OPTION:
            state="save it";
            break;
            // 放棄檔案儲存
            case JOptionPane.NO_OPTION:
            state="done";
            break;
            }
        }
        tonext(state,t,j);
    }
}
```

New pattern- Chain of Responsibility

If the state is “save it”, which means the user wants to save the file, then class “savehandler” will process it.

```
public class saveHandler extends fileHandler{
    public saveHandler(fileHandler next) {
        super(next);
    }

    @Override
    public void run(String state,JTextArea t,GUIfacade j) {
        if(state=="save it") {
            JFileChooser filechooser = new JFileChooser();
              int result = filechooser.showSaveDialog(j);
              if (result == JFileChooser.APPROVE_OPTION) {
                  try {
                      File file = filechooser.getSelectedFile();
                      FileWriter fw = new FileWriter(file);
                      BufferedWriter bw = new BufferedWriter(fw);
                      String text = t.getText();
                      fw.write(text);
                      fw.close();
                      bw.close();
                      state="done";
                  } catch (Exception ex) {
                      JOptionPane.showMessageDialog(j, "Oops! Mistakes happened when open the document");
                  }
              }
        }
        tonext(state,t,j);
    }
}
```

If the state is yes, it means the file was already saved, or if the state is done means we are processing the previous request, so the class “openHandler” and “exitHandler” will process the request to open an old file or to close the file.

```
public class openHandler extends fileHandler{
    public openHandler(fileHandler next) {
        super(next);
    }

    public void run(String state,JTextArea t,GUIfacade j) {
        if(state=="done" ||state=="yes") {
            JFileChooser filechooser = new JFileChooser();
              int result = filechooser.showOpenDialog(j);
              if (result == JFileChooser.APPROVE_OPTION) {
                  try {
                      File file = filechooser.getSelectedFile();
                      FileReader fr = new FileReader(file);
                      BufferedReader br = new BufferedReader(fr);
                      t.setText("");
                      String text;
                      while ((text = br.readLine()) != null) {
                          t.append(text);
                      }
                      fr.close();
                      br.close();
                  } catch (Exception ex) {
                      JOptionPane.showMessageDialog(j, "Something wrong happened when open the file!");
                  }
              }
        }
    }
}
```

Class “openHandler”

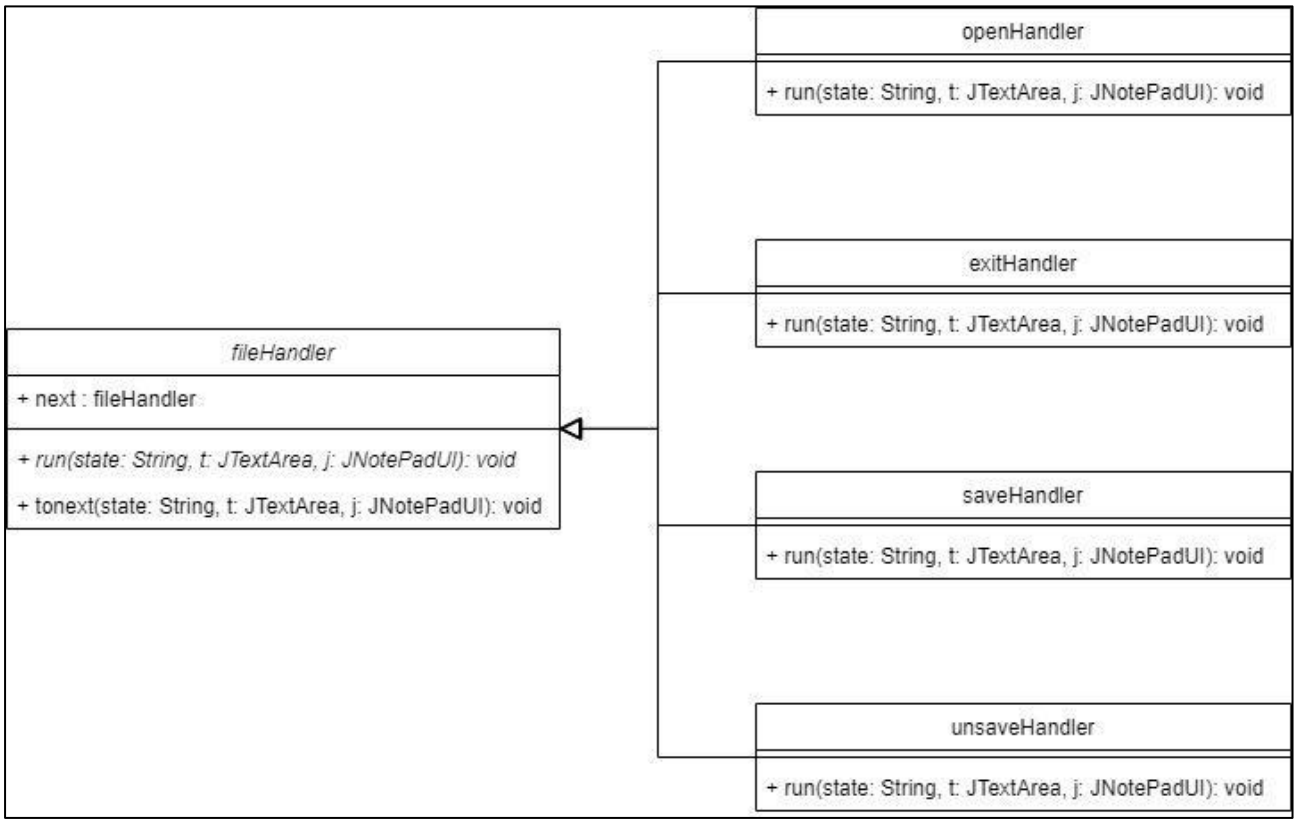
New pattern- Chain of Responsibility

```java
public class exitHandler extends fileHandler{
    public exitHandler(fileHandler next) {
        super(next);
    }

    @Override
    public void run(String state,JTextArea t,GUIfacade j) {
        if(state=="done"||state== "yes") {
            System.exit(0);

        }
    }
}
```

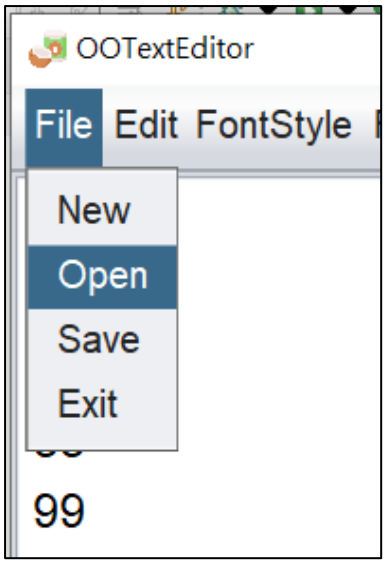
Class “exitHandler”

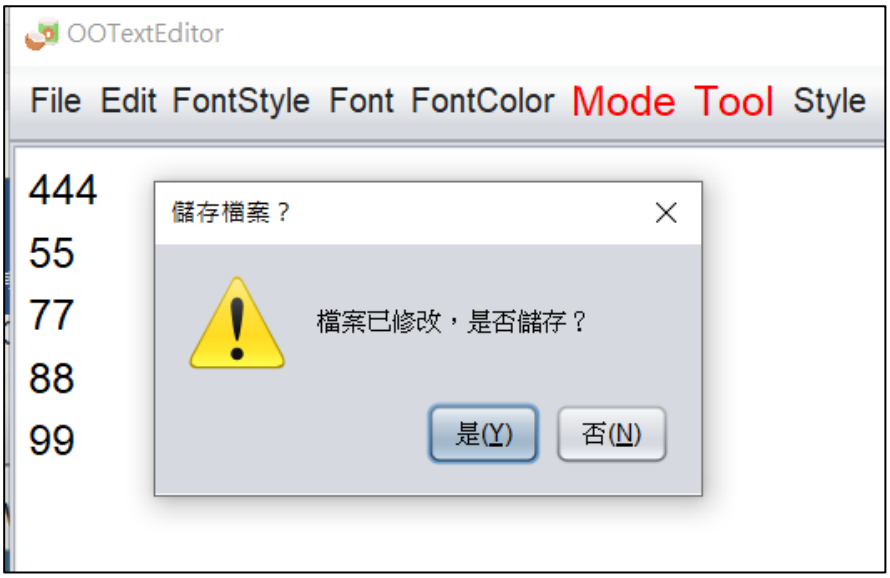
We have two chains in our text editor. It can check whether the file is saved when opening old files and closing the file.

```java
public fileHandler open=new unsaveHandler(new saveHandler(new openHandler(null)));
public fileHandler exit=new unsaveHandler(new saveHandler(new exitHandler(null)));
```

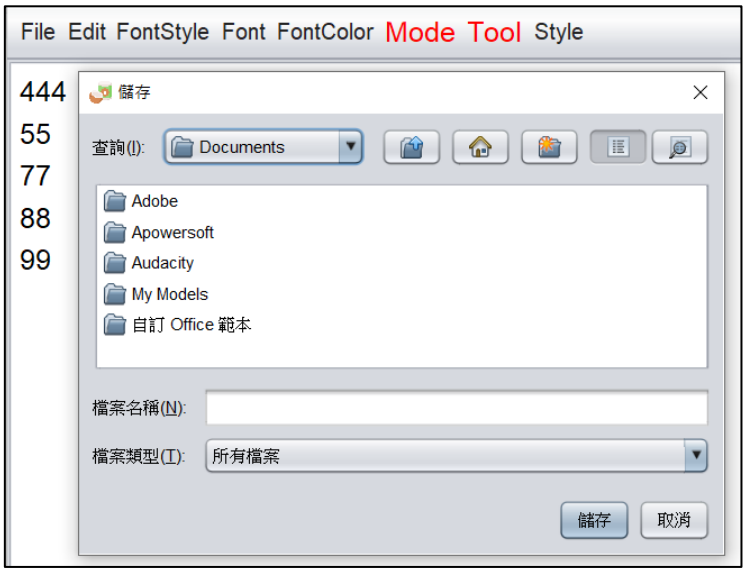


New pattern- Chain of Responsibility

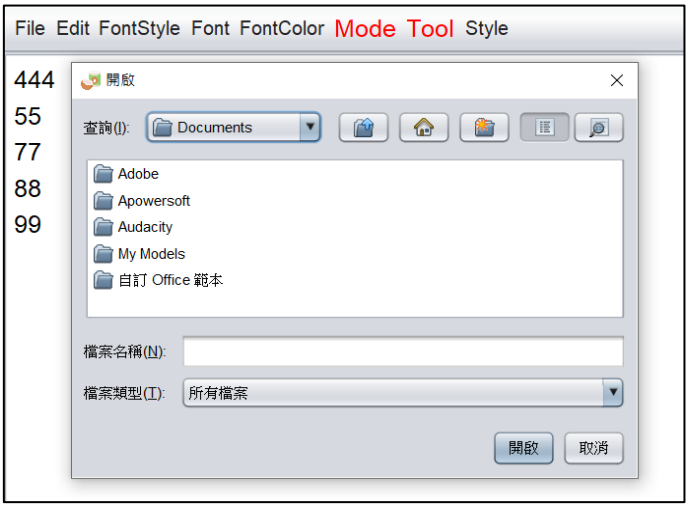




If we click open or exit while the file is not saved, it will show the dialog.



We can save the file.



And open an old file

New pattern-Decorator

Decorator

We use Decorate pattern to change the font style to Italic or Bold.

"General" class implements interface "Componet", and also defines objects that can be dynamically added by "DecoratorStyle" class.

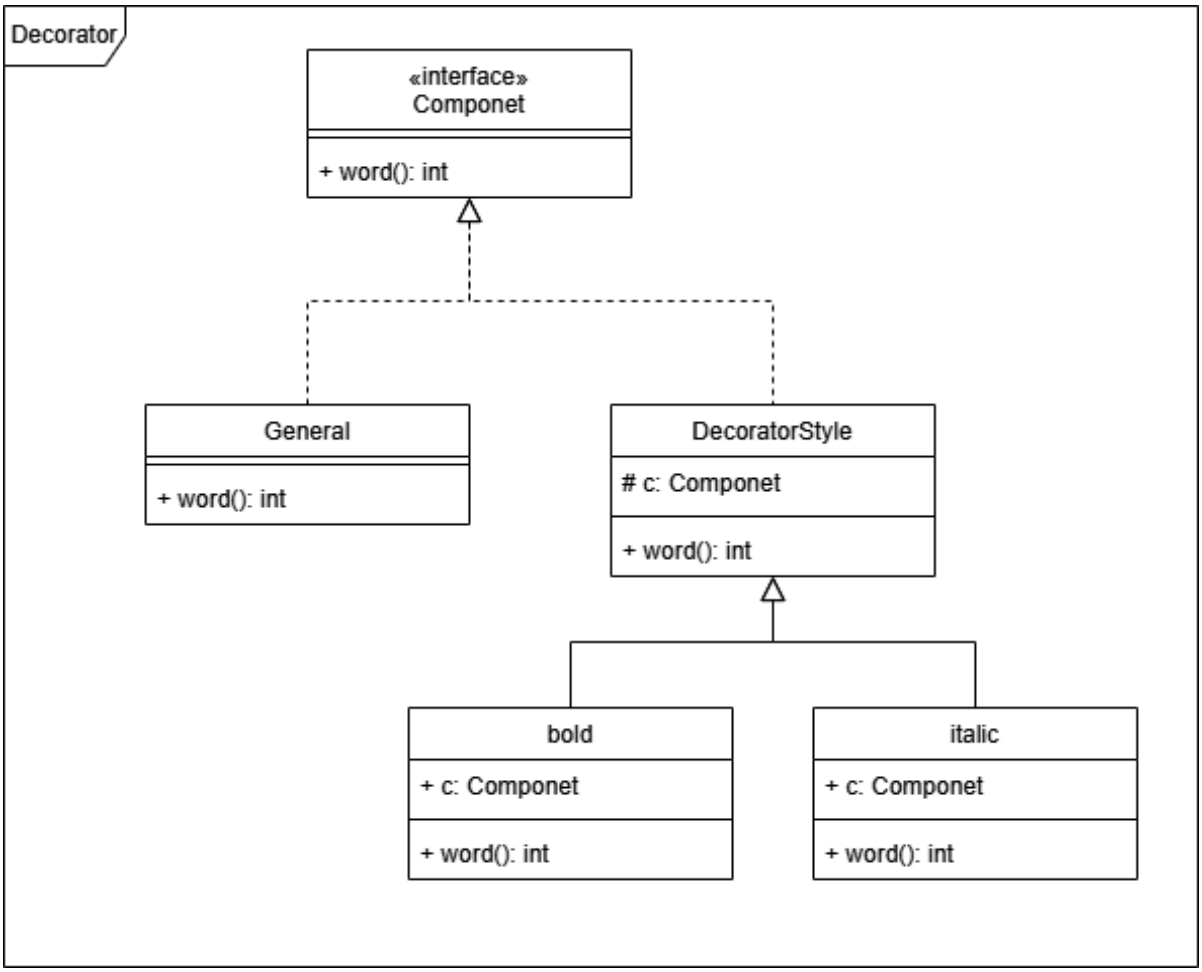
```
public class General implements Componet{
    public int word(){
        return Font.PLAIN;
    };
}
```

"italic" and "bold" class inheritance "DecoratorStyle" class, they add the style(Italic or Bold) to "General".

```
public class bold extends DecoratorStyle{
    Componet c;
    public bold(Componet c){
        this.c = c;
    }
    public int word(){
        return Font.BOLD + c.word();
    }
}
```

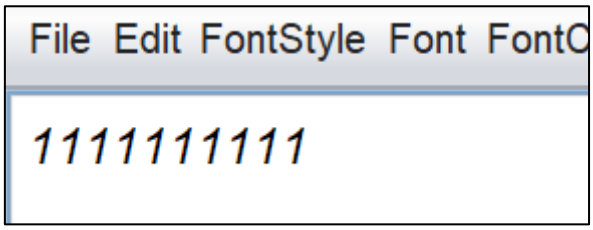
```
public class italic extends DecoratorStyle{
    Componet c;
        public italic(Componet c){
            this.c = c;
        }
        public int word(){
            return Font.ITALIC + c.word();
        }
}
```

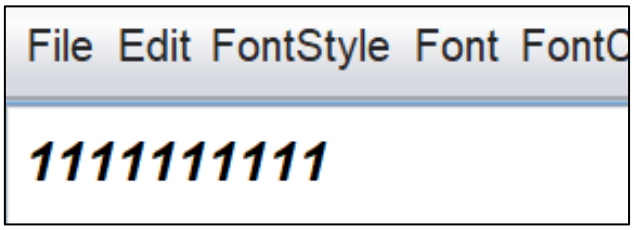
New pattern-Decorator



New pattern-Decorator

You can choose the font style: Italic or Bold in our texteditor. If you choose italic style first, then choose bold style, bold style will be superimposed on italic style.





New pattern

Observer

Observer pattern defines a one-to-many dependency among objects so that when one object changes state, all its dependents are notified and updated automatically.

In our system, the interface "Subject" has methods for adding and removing the observers, and a method for notifying the observer to update the status.

```
package Observer;

import javax.swing.JLabel;

public interface Subject {
    public void add(Observer o);
    public void remove(Observer o);
    public void notifying(JTextArea textarea,JLabel state);
}
```

In the interface "Observer", there is a method for updating the label's state.

```
package Observer;

import javax.swing.JLabel;

public interface Observer {
    public void update(JTextArea textarea, JLabel state);
}
```

New pattern-Observer

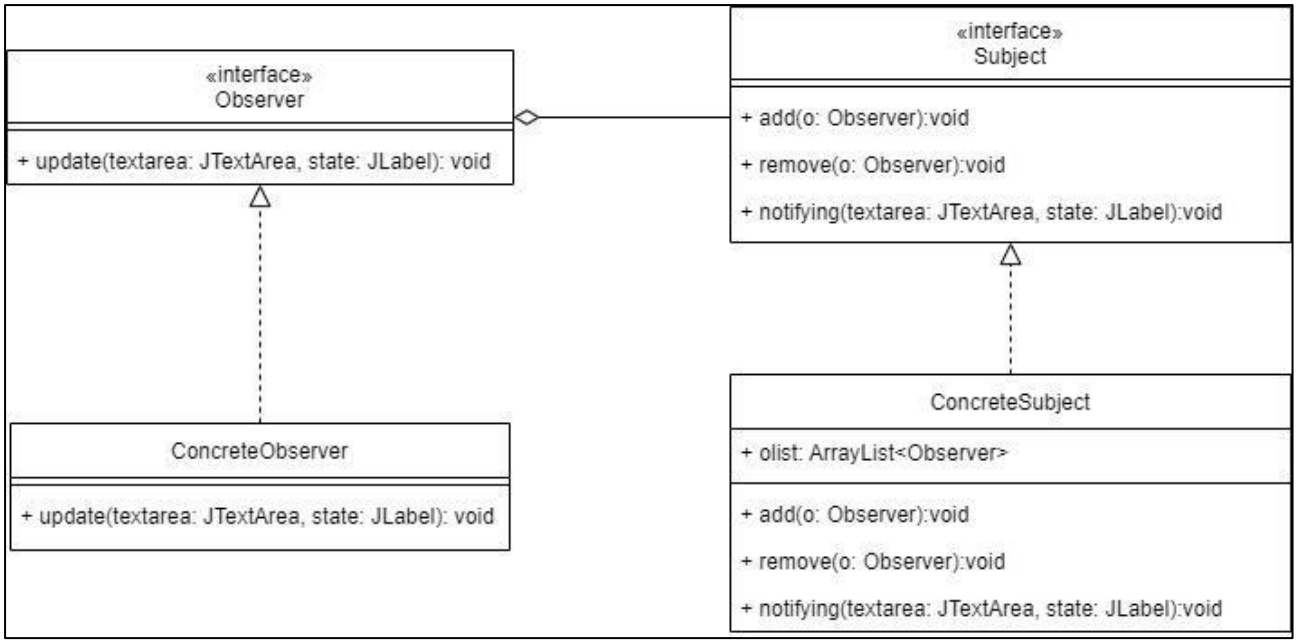
The class "ConcreteSubject" implements the interface "Subject", and we use an arraylist to store all observers. When the method notifying() is triggered, it will traverse the array list to notify each observer.

```
package Observer;

import java.util.ArrayList;

public class ConcreteSubject implements Subject{
    //用陣列儲存所有訂閱者
    ArrayList<Observer> olist=new ArrayList<Observer>();

    public void add(Observer o) {
        olist.add(o);
    }

    public void remove(Observer o) {
        olist.remove(o);
    }

    //更新狀態(送報紙)//通知訂閱者更新狀態了
    @Override
    public void notifying(JTextArea textarea, JLabel state) {
        for(Observer oo:olist) {
            oo.update(textarea,state);
        }
    }
}
```

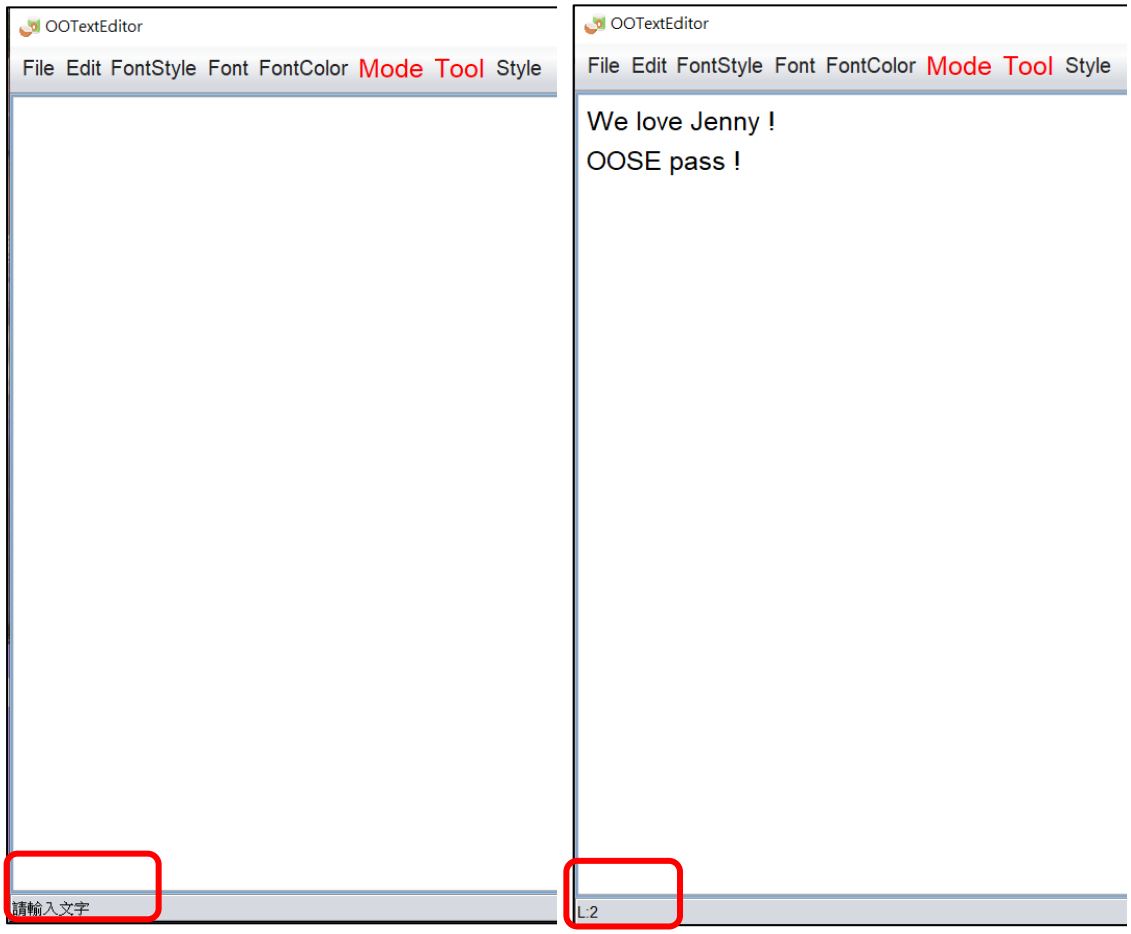
When the class "Concreteobserver" receives the notification from the method notifying(), the method update() updates the label's state.

```
package Observer;

import javax.swing.JLabel;
import javax.swing.JTextArea;

public class ConcreteObserver implements Observer{
        public void update(JTextArea textarea, JLabel state) {
            state.setText("L:"+textarea.getLineCount());
        }
}
```

New pattern-Observer



This is the application of the Observer pattern in our text editor. Observer pattern can count the line Rows.



New pattern

Flyweight

Flyweight pattern is used to support large numbers of fine-grained objects efficiently. Class “ConcreteCharactor” extends class “Charactor”. Method “getChar” gets the characters which the user inputs, and method “getUnicode” gets the ASCII of the characters. Finally, Method “draw” prints the ASCII of the characters.

```
public class ConcreteCharactor extends Charactor{
        private static int uniCode;
        ConcreteCharactor(char currentChar){
          uniCode = (int) currentChar;
        }
        public char getChar(){
            return (char)uniCode;
        }
        public int getUniCode() {
            return uniCode;
        }
        public void setUniCode(int uniCode) {
            ConcreteCharactor.uniCode = uniCode;
        }
        public void draw(){
            System.out.print(uniCode);
        }
}
```

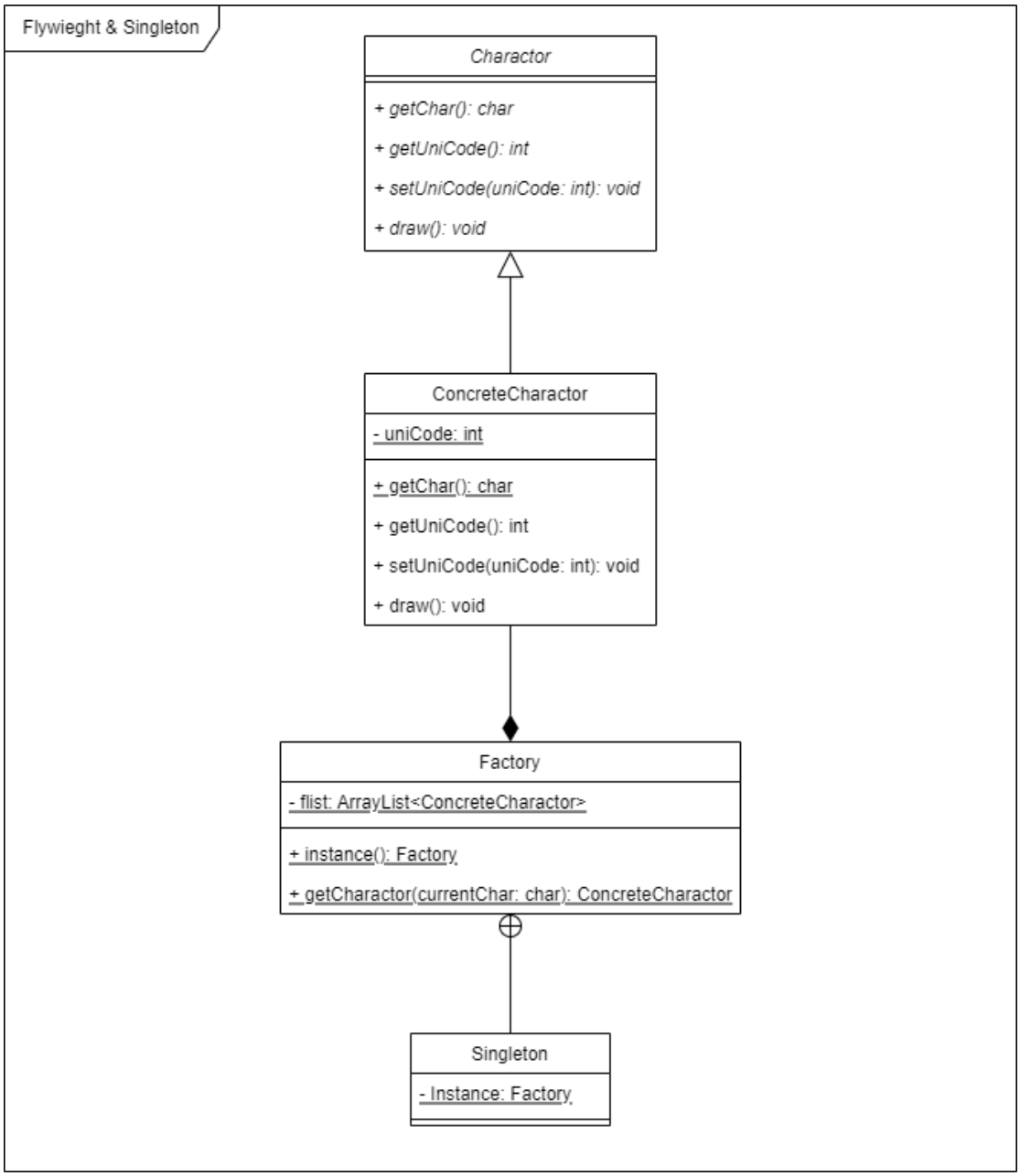
New pattern-Flyweight

We create an arraylist in the class “Factory” to store the component. Class “ConcreteCharactor” is shareable, it stores the intrinsic state, so we can use it to reduce the memory footprint and speed up our code.

Method “getCharacter” will check whether the character we get currently is the same with the characters in the list. If they are the same, return the character in the list; if not, that means it wasn't created before, so it will create a new one and add it into the list, then return it.

```
public class Factory {
    //創造元件的聚合池
        private static ArrayList<ConcreteCharactor> flist=new ArrayList<ConcreteCharactor>();
        //確保只創造一個Factory
        public class Singleton { ..

        //獲得內部資訊
        public static ConcreteCharactor getCharacter(char currentChar){
            //找看看list裡有沒有
            for (ConcreteCharactor character : flist) {
                if(ConcreteCharactor.getChar() == currentChar){
                    return character;//有就回傳
                }
            }
            //list裡沒有就創一個新的
            ConcreteCharactor character = new ConcreteCharactor(currentChar);
            flist.add(character);//並加進去
            return  character;//再回傳

        }
    }
```

New pattern-Flyweight



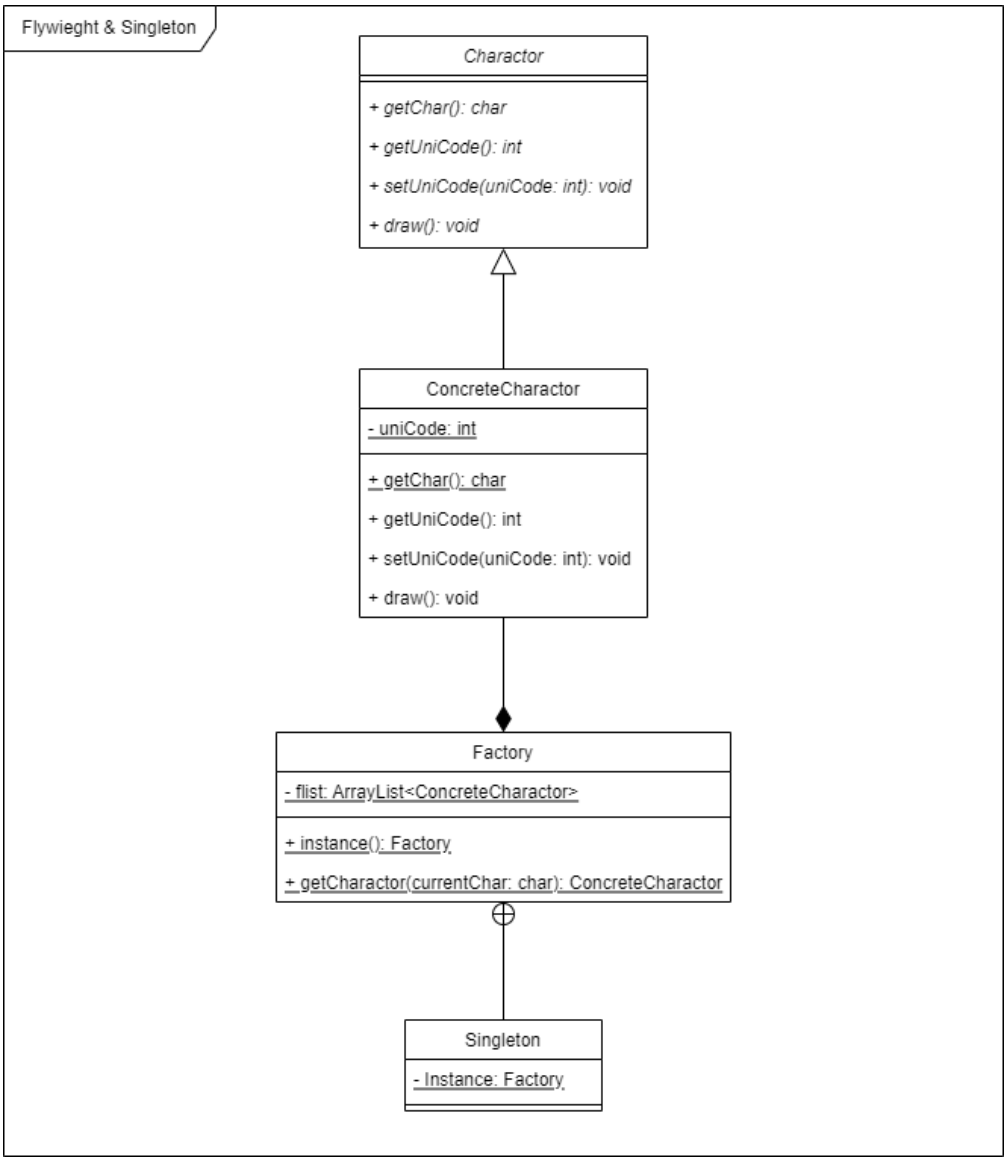
New pattern

Singleton

We use Singleton pattern to ensure we will only create one Factory.

```java
//確保只創造一個Factory
public class Singleton {
    private static Factory instance;
public static Factory instance() {
    if(instance==null) {
        synchronized(Singleton.class) {
            if(instance==null) instance=new Factory();
        }
    }
    return Singleton.instance;
     }
}
```

Class “Singleton” will new a pool to store the character we used before.



Old pattern

Abstract Factory

In our system, we use the class “EditorFactory” to create the MenuBar, Menu and MenuItem. Originally, we only had one ConcreteFactory defaultFactory to construct objects. Now we add a new ConcreteFactory specialFactory. When we need to add a special menuItem, we can use the specialFactory to create such a Menu Mode.

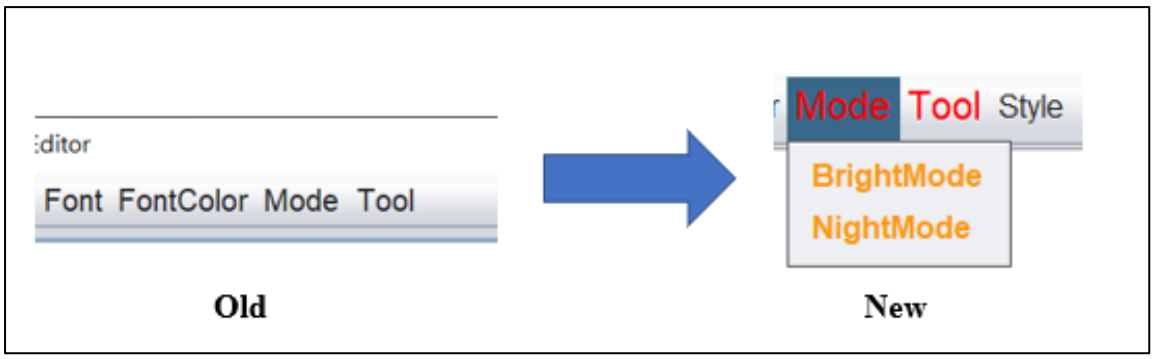
```
public interface EditorFactory {
    //每個object都是一個類別，也都有一個對應的方法建立
    public MenuBar createJMenuBar();
    public MenuItem creatJMenuItem();
    public Menu createJMenu();
}
```

```
public class defaultFactory implements EditorFactory{
    public MenuBar createJMenuBar() {
        return new defaultMenuBar();
    }
    public MenuItem creatJMenuItem() {
        return new defaultMenuItem();
    }
    public Menu createJMenu() {
        return new defaultMenu();
    }

}
```

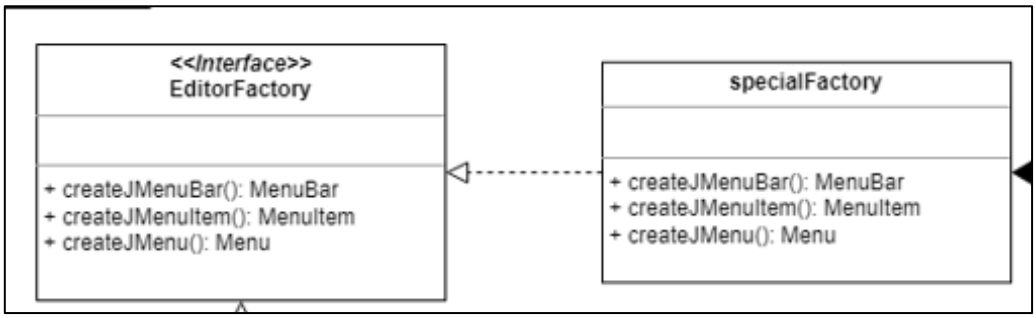
```
public class specialFactory implements EditorFactory {
    public MenuBar createJMenuBar() {
        return new specialMenuBar();
    }
    public MenuItem creatJMenuItem() {
        return new specialMenuItem();
    }
    public Menu createJMenu() {
        return new specialMenu();
    }
}
```

Old pattern- Abstract Factory

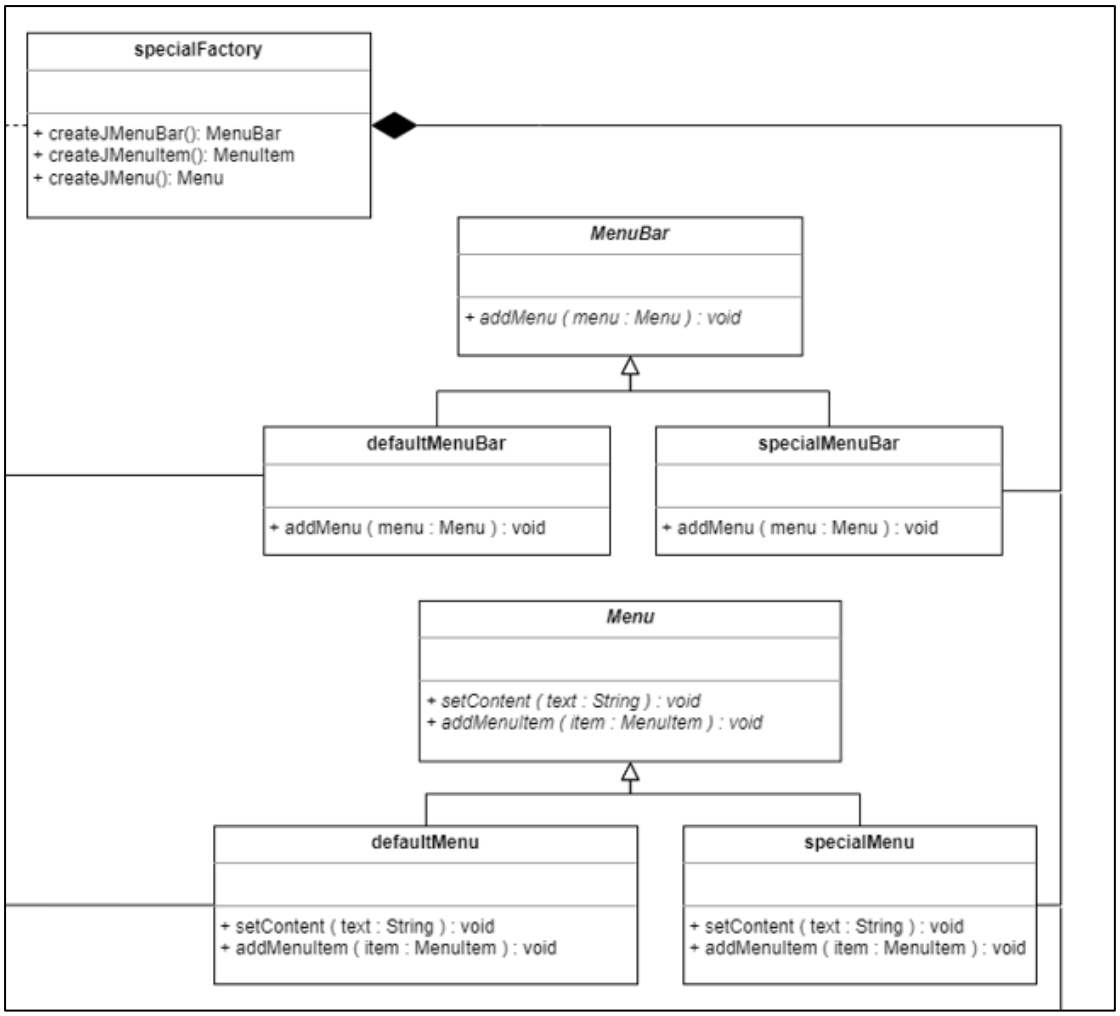
SpecialFactory can make different style Menu or MenuItem.(We made Menu Mode and Menu Tool using specialFactory)



SpecialFactory can provide specialMenu、specialMenuBar、specialItem.



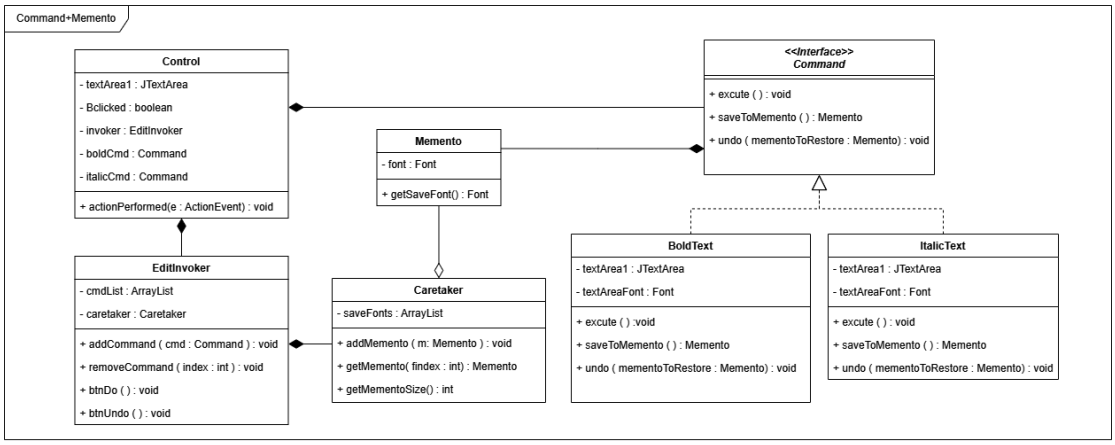
Old pattern- Abstract Factory



Old pattern

Command & Memento

We combine the command pattern and memento pattern. And this is the class diagram



Client will use the Control form the menuitem in menubar.

```java
public class Control implements MenuItemStrategy {

    private JTextArea textArea1 ;
    private boolean Bclicked =false;
    private EditInvoker invoker = new EditInvoker();
    private Command boldCmd ;
    private Command italicCmd ;

    public Control (GUIfacade gui) {
        textArea1  = gui.getTextArea();
        boldCmd = new BoldText(textArea1 );
        italicCmd = new ItalicText(textArea1 );
        invoker.addCommand(boldCmd);
        invoker.addCommand(italicCmd);
    }
    public void actionPerformed(ActionEvent e) {



        if(Bclicked == false) {
            invoker.btnDo();

            Bclicked = true;

        }else {
            invoker.btnUndo();
            Bclicked = false;
        }
    }
}
```

```java
package Command;

import java.util.ArrayList;

//client
public class EditInvoker {
    private ArrayList<Command> cmdList = new ArrayList<Command>();
    private Caretaker caretaker = new Caretaker();
    public void addCommand(Command cmd) {
        this.cmdList.add(cmd);
        System.out.println(cmdList.size());
    }

    public void removeCommand(int index) {
        cmdList.remove(index);
    }
    public void btnDo() {
        for(Command cmd : cmdList) {
        //執行前先儲存至caretaker
        caretaker.addMemento(cmd.saveToMemento());
        //執行
        cmd.execute();
        }

    }
    public void btnUndo() {
        //回復至先前儲存在caretaker的狀態
        for(int i = cmdList.size()-1 ; i >=0 ; i--) {
            Command cmd = cmdList.get(i);
            cmd.undo(caretaker.getMemento(i));

        }
    }
}
```

In constructor we new two commands which is boldcmd and italiccmd and we add these two command to invoker's arraylist, the cmdlist.

When we use the Control from menuitem in menubar:

1. invoker will do the btnDo() method
2. the boolean Bclicked is to make this menuitem like a switch when invoker do the btnDo() method.

Old pattern- Command & Memento

```java
    public void btnDo() {
        for(Command cmd : cmdList) {
        //執行前先儲存至caretaker
        caretaker.addMemento(cmd.saveToMemento());
        //執行
        cmd.execute();
        }
```

EditInvoker Class ▲

```java
package Command;
import java.awt.Font;


//用来儲存字體狀態
public class Memento {
    private Font font;

    public Memento(Font fontToSave) {
        font = fontToSave;
    }

    public Font getSaveFont() {
        return this.font;
    }

}
```

Memento Class ▲

```java
public class Caretaker {
    private ArrayList<Memento> saveFonts = new ArrayList<Memento>();

    public void addMemento(Memento m) {
        saveFonts.add(m);
    }

    public Memento getMemento(int findex) {
        return saveFonts.get(findex);
    }

    public int getMementoSize() {
        return saveFonts.size();
    }
}
```

```java
public class BoldText implements Command{
    private JTextArea textArea1 ;
    private Font textAreaFont;

    public BoldText(JTextArea jta) {
        this.textArea1 = jta;
    }

    public void execute() {
        if(textArea1.getFont().getStyle() == Font.PLAIN || textArea1.getFont().getStyle() == Font.BOLD ) {
            textAreaFont = new Font(textArea1.getFont().getName(),Font.BOLD,textArea1.getFont().getSize());
            textArea1.setFont(textAreaFont);
        }else {
            textAreaFont = new Font(textArea1.getFont().getName(),Font.BOLD +
                    textArea1.getFont().getStyle(),textArea1.getFont().getSize());
            textArea1.setFont(textAreaFont);
        }

    }

    public Memento saveToMemento() {
        return new Memento(textArea1.getFont());
    }

    public void undo(Memento memento) {
        textAreaFont = memento.getSaveFont();
        textArea1.setFont(textAreaFont);
    }

}
```

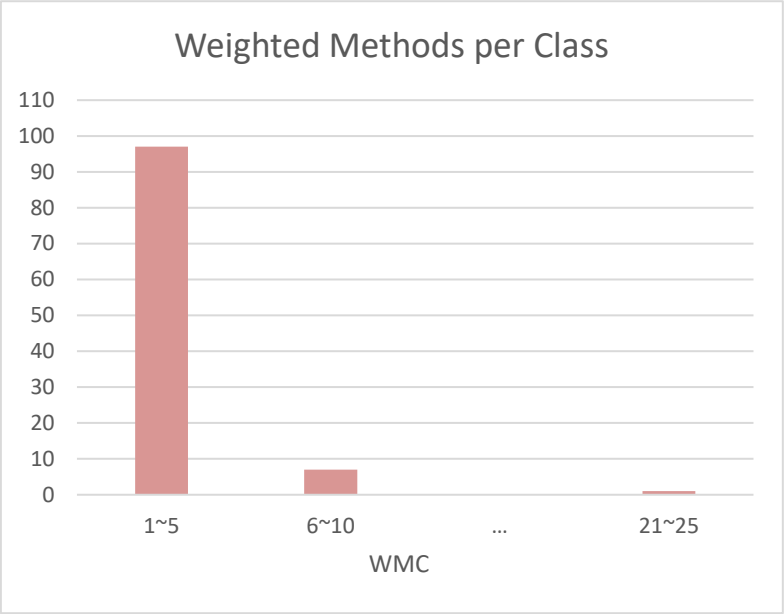
1. Command will do the saveToMemento() method then return the new Memento with the current Textarea's Font.
2. caretaker will add this memento in caretaker's arraylist
3. command do the execute() method , boldText command and Italic command change the textarea's font style here.

If we use the Control from menuitem in menubar again invoker will do the btnUndo() method from the last command to first command in cmdlist caretaker will return the memento in arraylist, then command use memento to do the undo() method to apply the previous font.

Quality Metrics

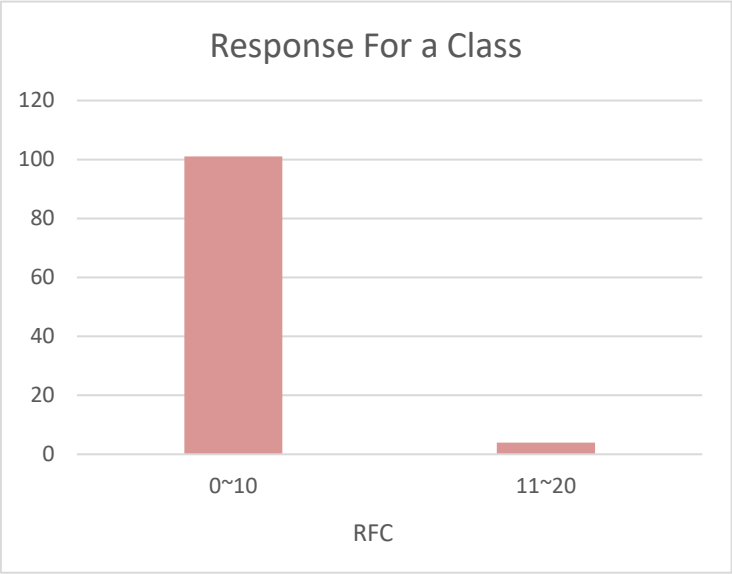
Weighted Methods per Class

Most of the class's WMC are very low in our new design. It means our program is not complexity and easy to maintenance and revise. But in the chart there has some class a bit high WMC. GUIFacade is the one of them, it integrate all the system functions together and provide UI to user.



Response For a Class

In this chart we can see most of class RFC is lower. This represent our code can be more easily to understand the methods then do test cases and debug. Class Form is high RFC because it has to execute the project and it has lots of methods.



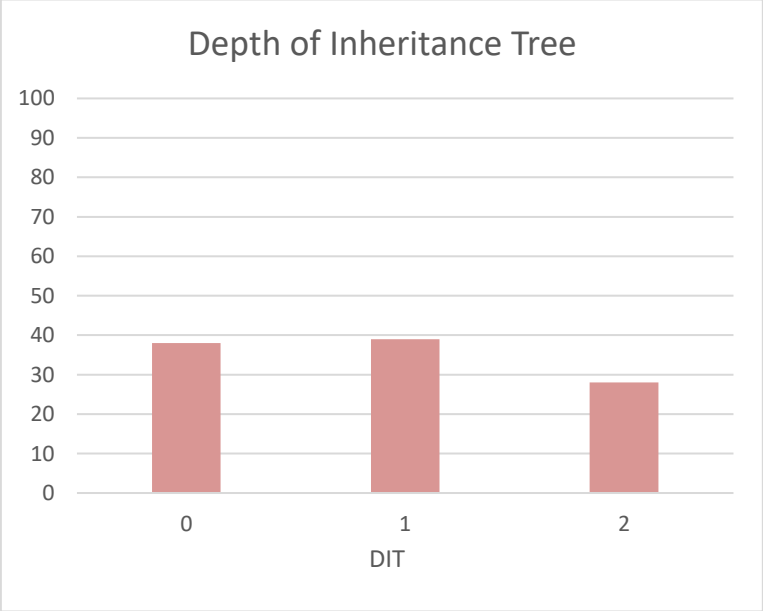
Quality Metrics

Coupling Between Object Classes

The following chart shows most of the CBO in our classes are low. But some class like defaultFactory and specialFactory are high. When user execute the code it will create and return the value, so the coupling will be more higher.

Depth of Inheritance Tree

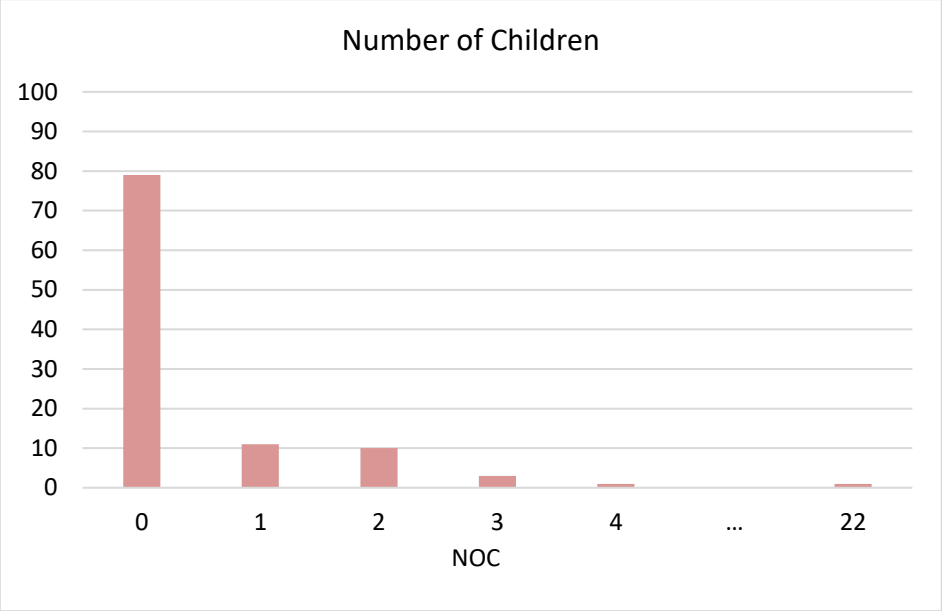
The lower DIT means that the code has less complexity. As the chart shows, you can see that most of classes DIT are not so deep. It means we can easily to reivse our code.



Quality Metrics

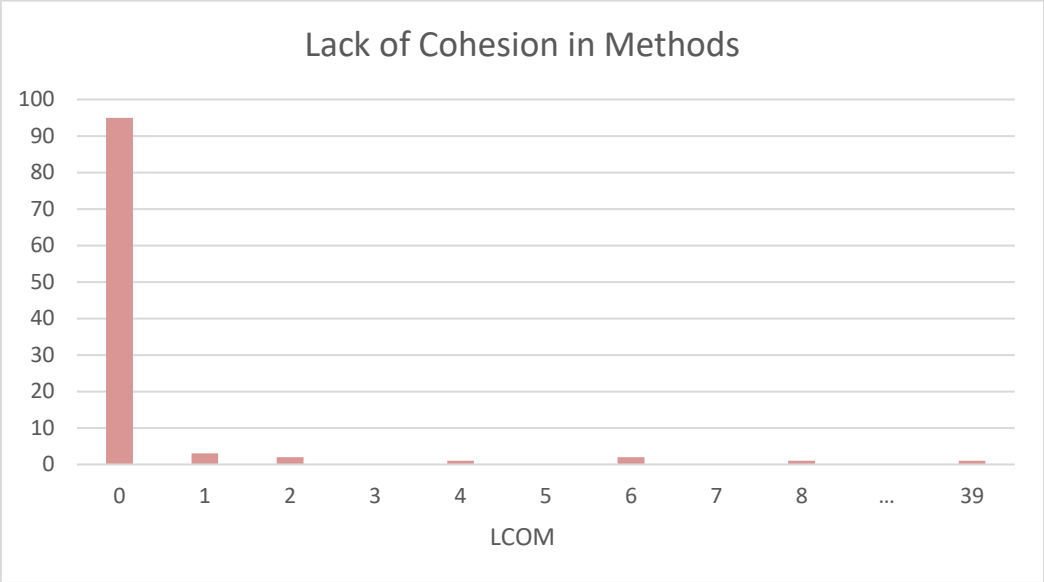
Number of Children

The NOC should be appropriate, so the class will easy to be reuse. If the NOC is too high, we may have to add more function in father and the will increase the coupling between the classes. It the chart the class MenuItemStrategy NOC is very high. Because the Strategy pattern has a lot of way can choose and you will know the action will be happened when you select.



Lack of Cohesion in Methods

If LCOM is high it means the relationship between are not stronge enough, so you can add more subclass to decrease the LCOM, that each class has more stronger cohesion. The class GUIFacade has very high cohesion, because it do many thing to integrate all the system functions.



Black Box Testing

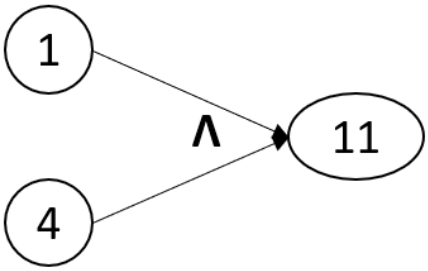
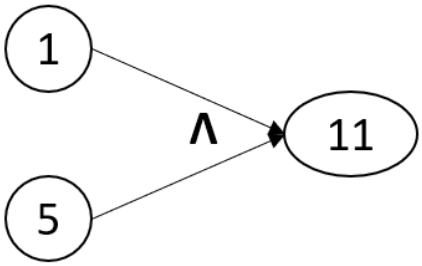
Black&White Box Testing

BLACK BOX TESTING

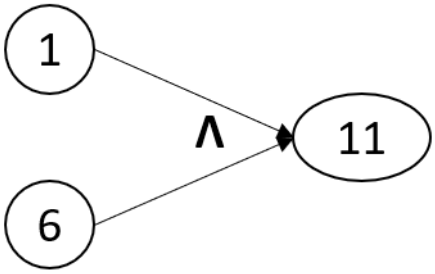
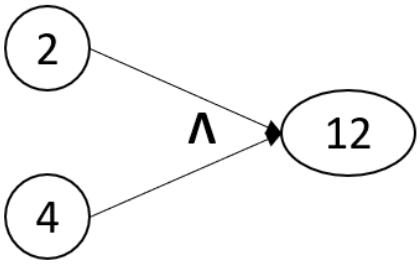
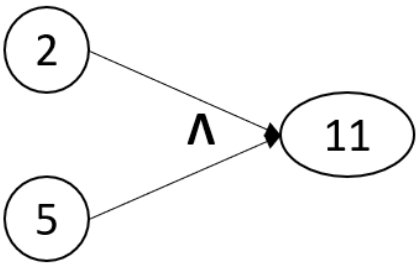
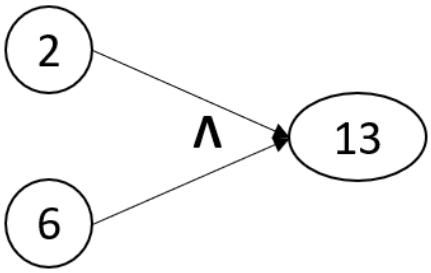
Cause-Effect Analysis of Font

We use the fonts to do causal analysis. Because not every font can match a different language. In our text editor, Arial and Impact are fonts exclusively for English. DFKai-SB, PMingLiU and Microsoft JhengHei are universal fonts. Caveat will determine whether it is an English font. If so, change it to that font. If not, change to the default font.

Input	Output
1. Text=English 2. Text=Chinese 3. Text=Korean 4. Font=Arial、Impact 5. Font=DFKai-SB、PMingLiU、Microsoft JhengHei 6. Font=Caveat	11. Change to the correct font 12. Changed to not display (□) 13. Change to default font

Graph	Interpretation
	Causes：1. Text=English And(∧) 4. Font=Arial、Impact Effect： 11. Change to the correct font
	Causes：1. Text=English And(∧) 5. Font=DFKai-SB、PMingLiU Effect： 11. Change to the correct font

Black Box Testing

Graph	Interpretation
	Causes：1. Text=English And(**Λ**) 6. Font=Caveat Effect： 11. Change to the correct font
	Causes：2. Text=Chinese And(**Λ**) 4. Font=Arial、Impact Effect： 12. Changed to not display(□)
	Causes：2. Text=Chinese And(**Λ**) 5. Font=DFKai-SB、 PMingLiU Effect： 11. Change to the correct font
	Causes：2. Text=Chinese And(**Λ**) 6. Font=Caveat Effect： 13. Change to default font
	Causes：3. Text=Korean Effect： 12. Changed to not display(□)

	1	2	3	4	5	6	7
Text	English	English	English	Chinese	Chinese	Chinese	Korean
Font	Arial Impact	DFKai-SB PMingLiU Microsoft JhengHei	Caveat	Arial Impact	DFKai-SB PMingLiU Microsoft JhengHei	Caveat	-
Count	1	1	1	1	1	1	3

Black Box Testing

Result	11	11	11	12	11	13	12

11. Change to the correct font
12. Changed to not display(□)
13. Change to default font

Black Box Testing

Boundary Value Analysis of FontColor(RGB)

We use the font's color to do boundary value analysis. Because there are three parameters in the RGB color model: red, green, and blue, and the range of each parameter is 0 to 255. In our text editor, if the parameter setting is lower than 0, no input is allowed, if the parameter setting is greater than 255, it will be automatically changed to 255.

Test Cases	
Red{0,255}	-1, 0, 1, 254, 255, 256
Green{0,255}	-1, 0, 1, 254, 255, 256
Blue{0,255}	-1, 0, 1, 254, 255, 256

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
1	Red=-1 Green=50 Blue=50	The red value cannot be changed	The red value cannot be changed	EO=AO	Qualified
2	Red=0 Green=50 Blue=50	Change the font color to #0032FF	Change the font color to #0032FF	EO=AO	Qualified
3	Red=1 Green=50 Blue=50	Change the font color to #003232	Change the font color to #003232	EO=AO	Qualified
4	Red=254 Green=50 Blue=50	Change the font color to #FE32FF	Change the font color to #FE32FF	EO=AO	Qualified
5	Red=255 Green=50 Blue=50	Change the font color to #FF32FF	Change the font color to #FF32FF	EO=AO	Qualified
6	Red=256 Green=50 Blue=50	The red value is automatically changed to 255. Change the font color to #FF32FF	The red value is automatically changed to 255. Change the font color to #FF32FF	EO=AO	Qualified
7	Red=50 Green=-1 Blue=50	The green value cannot be changed	The green value cannot be changed	EO=AO	Qualified

Black Box Testing

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
8	Red=50 Green=0 Blue=50	Change the font color to #320032	Change the font color to #320032	EO=AO	Qualified
9	Red=50 Green=1 Blue=50	Change the font color to #320132	Change the font color to #320132	EO=AO	Qualified
10	Red=50 Green=254 Blue=50	Change the font color to #32FE32	Change the font color to #32FE32	EO=AO	Qualified
11	Red=50 Green=255 Blue=50	Change the font color to #32FF32	Change the font color to #32FF32	EO=AO	Qualified
12	Red=50 Green=256 Blue=50	The green value is automatically changed to 255. Change the font color to #32FF32	The green value is automatically changed to 255. Change the font color to #32FF32	EO=AO	Qualified
13	Red=50 Green=50 Blue=-1	The blue value cannot be changed	The blue value cannot be changed	EO=AO	Qualified
14	Red=50 Green=50 Blue=0	Change the font color to #323200	Change the font color to #323200	EO=AO	Qualified
15	Red=50 Green=50 Blue=1	Change the font color to #323201	Change the font color to #323201	EO=AO	Qualified
16	Red=50 Green=50 Blue=254	Change the font color to #3232FE	Change the font color to #3232FE	EO=AO	Qualified
17	Red=50 Green=50 Blue=255	Change the font color to #3232FF	Change the font color to #3232FF	EO=AO	Qualified

Black Box Testing

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
18	Red=50 Green=50 Blue=256	The blue value is automatically changed to 255. Change the font color to #3232FF	The blue value is automatically changed to 255. Change the font color to #3232FF	EO=AO	Qualified

Black Box Testing

Boundary Value Analysis of FontColor(RGB)

We use the font's color to do boundary value analysis. Because there are three parameters in the RGB color model: red, green, and blue, and the range of each parameter is 0 to 255. In our text editor, if the parameter setting is lower than 0, no input is allowed, if the parameter setting is greater than 255, it will be automatically changed to 255.

Test Cases	
Red{0,255}	-1, 0, 1, 254, 255, 256
Green{0,255}	-1, 0, 1, 254, 255, 256
Blue{0,255}	-1, 0, 1, 254, 255, 256

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
1	Red=-1 Green=50 Blue=50	The red value cannot be changed	The red value cannot be changed	EO=AO	Qualified
2	Red=0 Green=50 Blue=50	Change the font color to #0032FF	Change the font color to #0032FF	EO=AO	Qualified
3	Red=1 Green=50 Blue=50	Change the font color to #003232	Change the font color to #003232	EO=AO	Qualified
4	Red=254 Green=50 Blue=50	Change the font color to #FE32FF	Change the font color to #FE32FF	EO=AO	Qualified
5	Red=255 Green=50 Blue=50	Change the font color to #FF32FF	Change the font color to #FF32FF	EO=AO	Qualified
6	Red=256 Green=50 Blue=50	The red value is automatically changed to 255. Change the font color to #FF32FF	The red value is automatically changed to 255. Change the font color to #FF32FF	EO=AO	Qualified
7	Red=50 Green=-1 Blue=50	The green value cannot be changed	The green value cannot be changed	EO=AO	Qualified

Black Box Testing

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
8	Red=50 Green=0 Blue=50	Change the font color to #320032	Change the font color to #320032	EO=AO	Qualified
9	Red=50 Green=1 Blue=50	Change the font color to #320132	Change the font color to #320132	EO=AO	Qualified
10	Red=50 Green=254 Blue=50	Change the font color to #32FE32	Change the font color to #32FE32	EO=AO	Qualified
11	Red=50 Green=255 Blue=50	Change the font color to #32FF32	Change the font color to #32FF32	EO=AO	Qualified
12	Red=50 Green=256 Blue=50	The green value is automatically changed to 255. Change the font color to #32FF32	The green value is automatically changed to 255. Change the font color to #32FF32	EO=AO	Qualified
13	Red=50 Green=50 Blue=-1	The blue value cannot be changed	The blue value cannot be changed	EO=AO	Qualified
14	Red=50 Green=50 Blue=0	Change the font color to #323200	Change the font color to #323200	EO=AO	Qualified
15	Red=50 Green=50 Blue=1	Change the font color to #323201	Change the font color to #323201	EO=AO	Qualified
16	Red=50 Green=50 Blue=254	Change the font color to #3232FE	Change the font color to #3232FE	EO=AO	Qualified
17	Red=50 Green=50 Blue=255	Change the font color to #3232FF	Change the font color to #3232FF	EO=AO	Qualified

Black Box Testing

Case#	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
18	Red=50 Green=50 Blue=256	The blue value is automatically changed to 255. Change the font color to #3232FF	The blue value is automatically changed to 255. Change the font color to #3232FF	EO=AO	Qualified

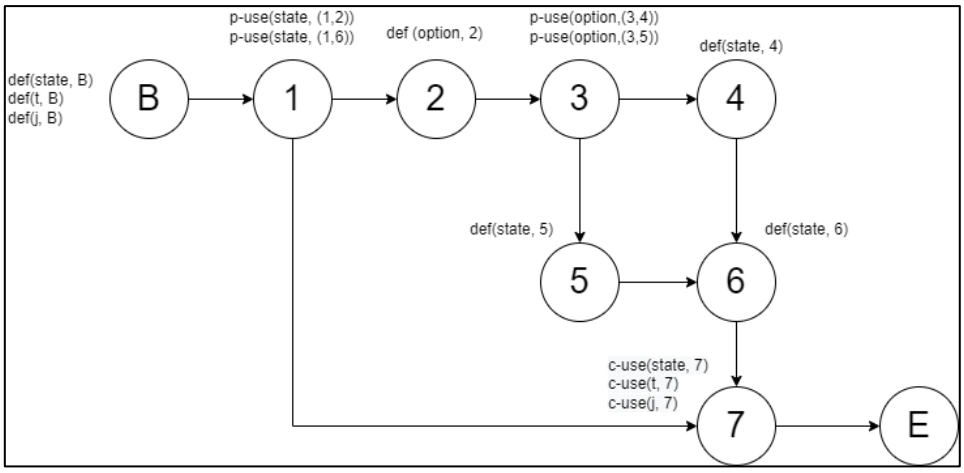
White Box Testing

unsaveHandler

run()

```
package ChainOfResponsibility;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import Form.GUIfacade;

public class unsaveHandler extends fileHandler {

    public unsaveHandler(fileHandler next) {
        super(next);
    }
B public void run(String state,JTextArea t,GUIfacade j) {
1       if(state=="no") {
2           int option =JOptionPane.showConfirmDialog(
            null,"檔案已修改，是否儲存？",
            "儲存檔案？",JOptionPane.YES_NO_OPTION,
            JOptionPane.WARNING_MESSAGE,null);

3           switch(option){
            // 確認檔案儲存
4           case JOptionPane.YES_OPTION:
            state="save it";
            break;
            // 放棄檔案儲存
5           case JOptionPane.NO_OPTION:
            state="done";
            break;
            }
6           this.state=state;
        }
7       tonext(state,t,j);
    }
}
```

White Box Testing



Basis Paths

Path1：B→1→2→3→5→6→7→E

When opening other files, the current file has not been saved yet. After exiting the window, choose not to save.

Path2：B→1→2→3→4→6→7→E

When opening other files, the current file has not been saved. After popping out the window, choose to save.

Path3：B→1→7→E

When opening other files, the current file has been saved

Cyclomatic Complexity

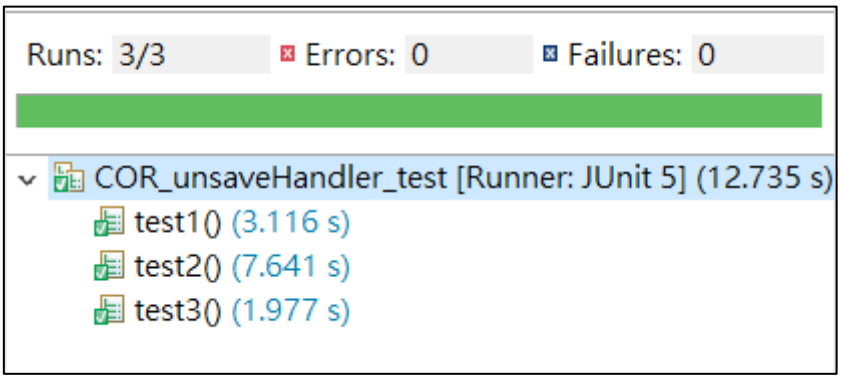
1. Number of closed regions plus one:2+1=3
2. Number of nodes and edges:10-9+2=3
3. Number of atomic binary conditions plus one:2+1=3

Node coverage	100%
Edge coverage	100%
Basis path coverage	100%

Variable	c-use	p-use
state	(B,7)	(B,(1,2))、(B,(1,6))
t	(B,7)	-
j	(B,7)	-
option	-	(2,(3,4))、(2,(3,5))

White Box Testing

```java
package ChainOfResponsibility;
import javax.swing.*;
class COR_unsaveHandler_test {
    static GUIfacade fa;
    static JTextArea aa;
    static fileHandler a;
    @BeforeEach
    void setUp() {
    fa=new GUIfacade();
    fa.initComponents();
    fa.setVisible(true);
    aa=fa.textArea1;
    fa.initComponents();
    a =new unsaveHandler(new saveHandler(new openHandler(null)));

    }
    @Test
    void test1() {
        //於開啟其他檔案時發現現有檔案尚未儲存
        a.run("no", aa, fa);
        //於要儲存時選擇不儲存
        assertEquals("done",a.state);
    }
    @Test
    void test2() {
        //於開啟其他檔案時發現現有檔案尚未儲存
        a.run("no", aa, fa);
        //於要儲存時選擇儲存
        assertEquals("save it",a.state);
    }
    @Test
    void test3() {
        //於開啟其他檔案時現有檔案已儲存
        a.run("yes", aa, fa);
        assertNull(a.state);
    }
}
```

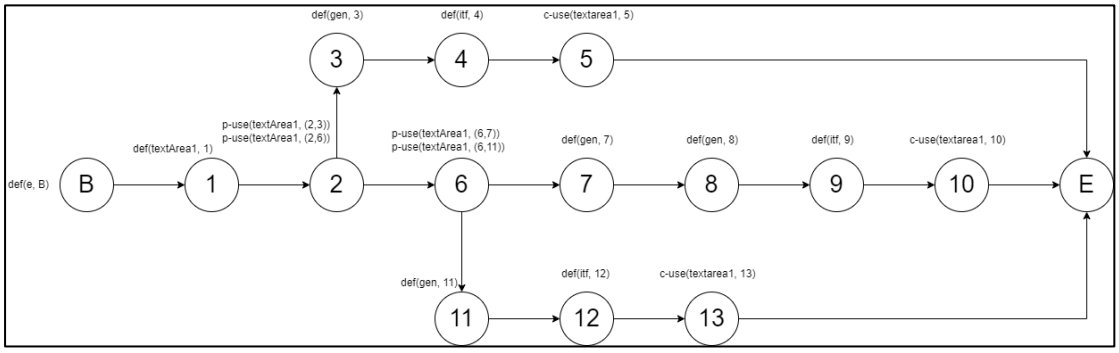


White Box Testing

boldFont

actionPerformed()

```
package Strategy;
import java.awt.Font;

public class boldFont implements MenuItemStrategy {
    private JTextArea textArea2;
    public Componet gen;
    public italicFont itf;
    public boldFont(JTextArea jta , Componet g,italicFont itf) {
        textArea2 = jta;
        gen = g;
        this.itf = itf;
    }

B public void actionPerformed(ActionEvent e) {
1       JTextArea textArea1=textArea2;
2       if(textArea1.getFont().getStyle()==Font.BOLD) {
3           gen = new General();
4           itf.gen=gen;
5           textArea1.setFont(new Font(textArea1.getFont().getName(),gen.word(),textArea1.getFont().getSize()));
        }
6       else if(textArea1.getFont().getStyle()==Font.BOLD+Font.ITALIC) {
7           gen = new General();
8           gen = new Decorator.italic(gen);
9           itf.gen=gen;
10          textArea1.setFont(new Font(textArea1.getFont().getName(),gen.word(),textArea1.getFont().getSize()));
        }
        else {
11          gen = new Decorator.bold(gen);
12          itf.gen=gen;
13          textArea1.setFont(new Font(textArea1.getFont().getName(),gen.word(),textArea1.getFont().getSize()));
            }
        }
}
```



Basis Paths

Path1：B→1→2→3→4→5→E

If the current font is bold, cancel the bold.

Path2：B→1→2→6→7→8→9→10→E

If the current font is bold and italic, cancel the bold and leave the italic.

Path3：B→1→2→6→11→12→13→E

If the current font is the original state, it will become bold

Cyclomatic Complexity

1. Number of closed regions plus one:2+1=3
2. Number of nodes and edges:16-15+2=3
3. Number of atomic binary conditions plus one:2+1=3

White Box Testing

Node coverage	100%
Edge coverage	100%
Basis path coverage	100%

Variable	c-use	p-use
gen	-	-
itf	-	-
textArea1	(B,5)、(B,10)、(B,13)	(1,(2,3))、(1,(2,6))、(1,(6,7))、(1,(6,11))

Invocation chains

Number of Methods	1	2	3	4	5	6
Number of chains	141	62	18	31	7	0

14+5+10+10+5+13+5=(2)
5+6+2+5=(3)

◆ The GUIfacade activity is used to control the text editor. The reason why this chain is the longest one is that when clicking every single button for each text editor, it will trigger a series of methods. Therefore, the chain is the longest.

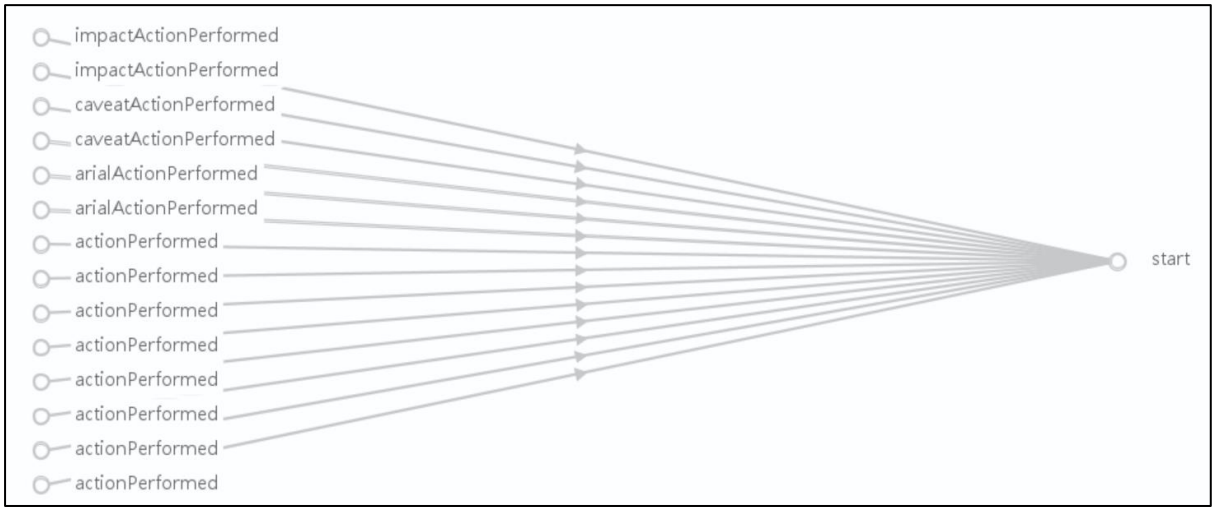
◆ For example, The initComponents method will call setToolMenu to choose the different type of the method. After that, the spellCheckActionPerformed will call the length or check the length such as hasNext or gerText.

This picture down below does not cause any invocation chains.

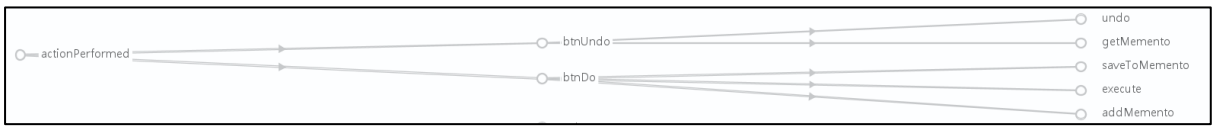
Invocation chains

Those are the methods which cause an invocation chain. (14)

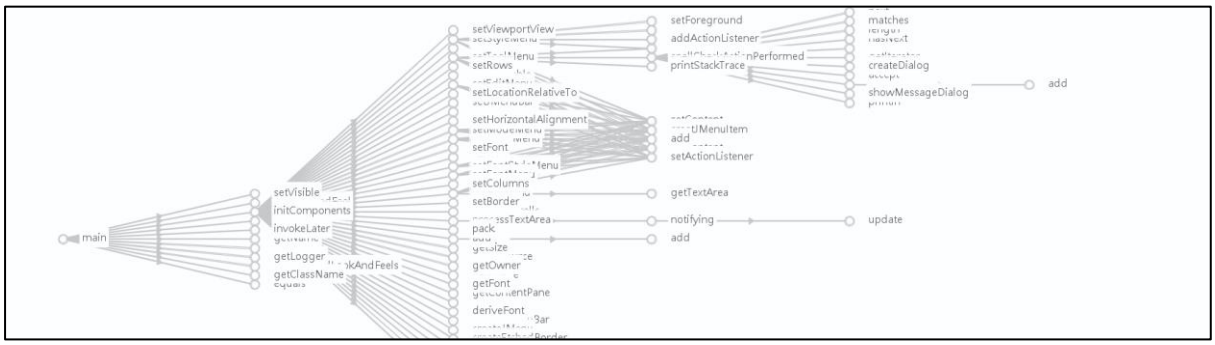
example: impactActionPerformed -> start => 1



Those are the methods which cause two invocation chains.
example: actionPerformed -> btnUndo -> undo => 2

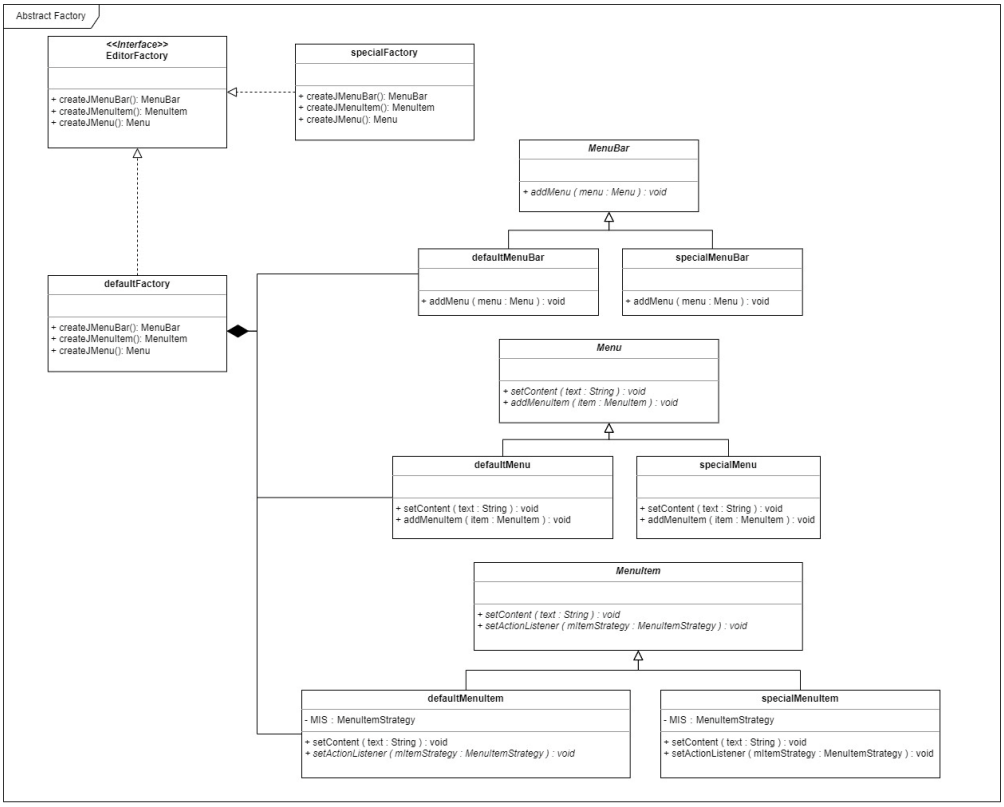


Those are the methods which cause three and four invocation chains.
example:
main -> initComponents -> setStyleMenu ->setFont =>3
main -> initComponents -> setToolMenu-> spellCheckActionPerformed-> next =>4



The relationship of our design

Abstract Factory



The number of class: 12

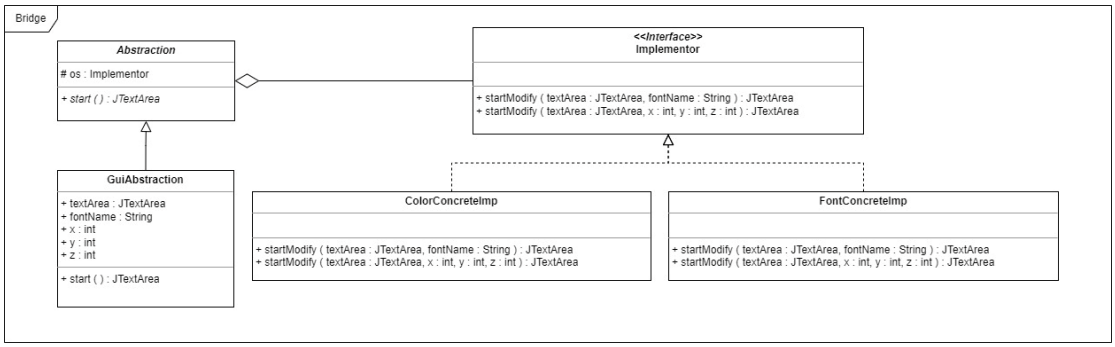
Composition: 3
Class defaultMenuBar is a part of class defaultFactory.
Class defaultMenu is a part of class defaultFactory.
Class defaultMenuItem is a part of class defaultFactory.

Inheritance: 6
Class defaultMenuBar is a subclass of MenuBar.
Class specialMenuBar is a subclass of MenuBar.
Class defaultMenu is a subclass of Menu.
Class specialMenu is a subclass of Menu.
Class defaultMenuItem is a subclass of MenuItem.
Class specialMenuItem is a subclass of MenuItem.

Implement: 2
Class specialFactory implements EditorFactory.
Class defaultFactory implements EditorFactory.

The relationship of our design

Bridge



The number of class: 5

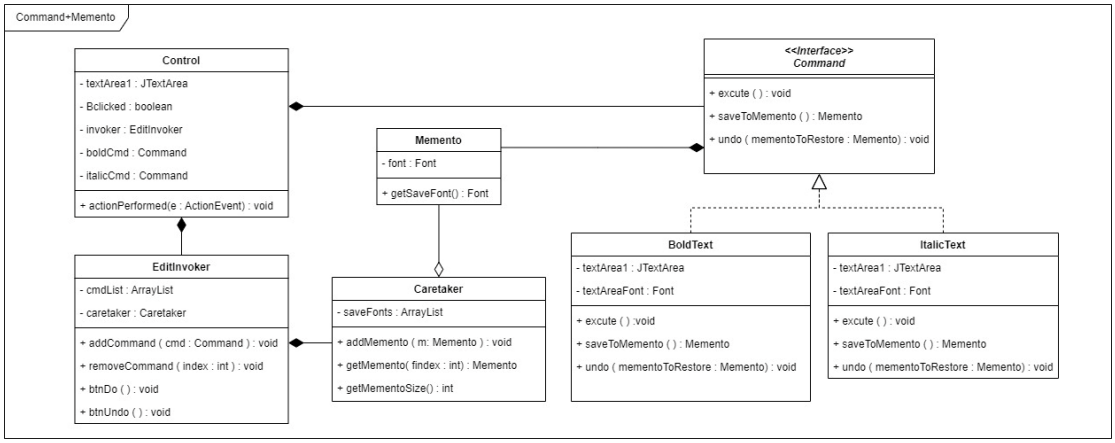
Aggregation: 1
Class Abstraction owns class Implementor.

Inheritance: 1
Class GuiAbstraction is a subclass of class Abstraction.

Implement: 2
Class ColorConcreteImp implements class Implementor.
Class FontConcreteImp implements class Implementor.

The relationship of our design

Command-Memento



The number of class: 7

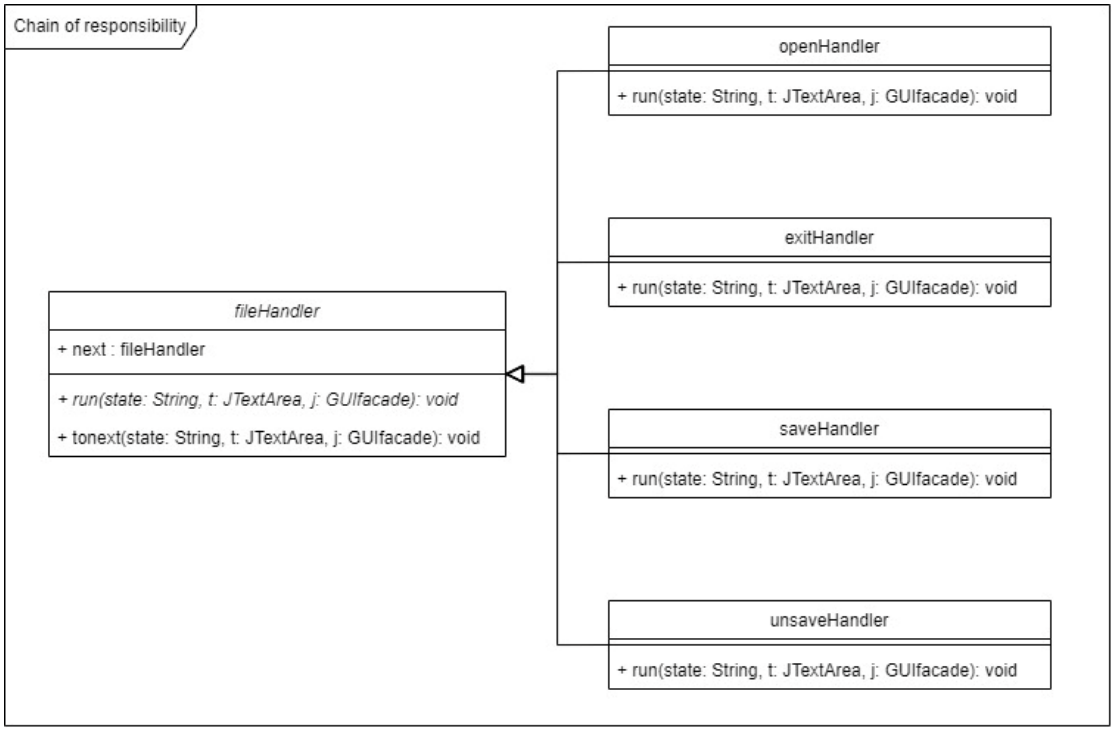
Aggregation: 1
Class Caretaker owns class Memento.

Composition: 4
Class Command is a part of class Control.
Class EditInvoker is a part of class Control.
Class Memento is a part of class Command.
Class Caretaker is a part of class EditInvoker.

Implement: 2
Class BoldText implements class Command.
Class ItalicText implements class Command.

The relationship of our design

Chain of responsibility



The number of class: 5

Inheritance: 4

Class openHandler is a subclass of class fileHandler.

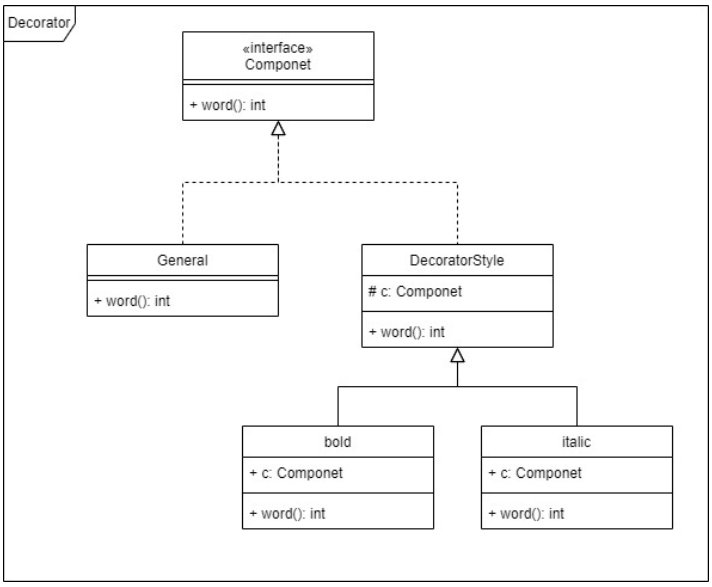
Class exitHandler is a subclass of class fileHandler.

Class saveHandler is a subclass of class fileHandler.

Class unsaveHandler is a subclass of class fileHandler.

The relationship of our design

Decorator



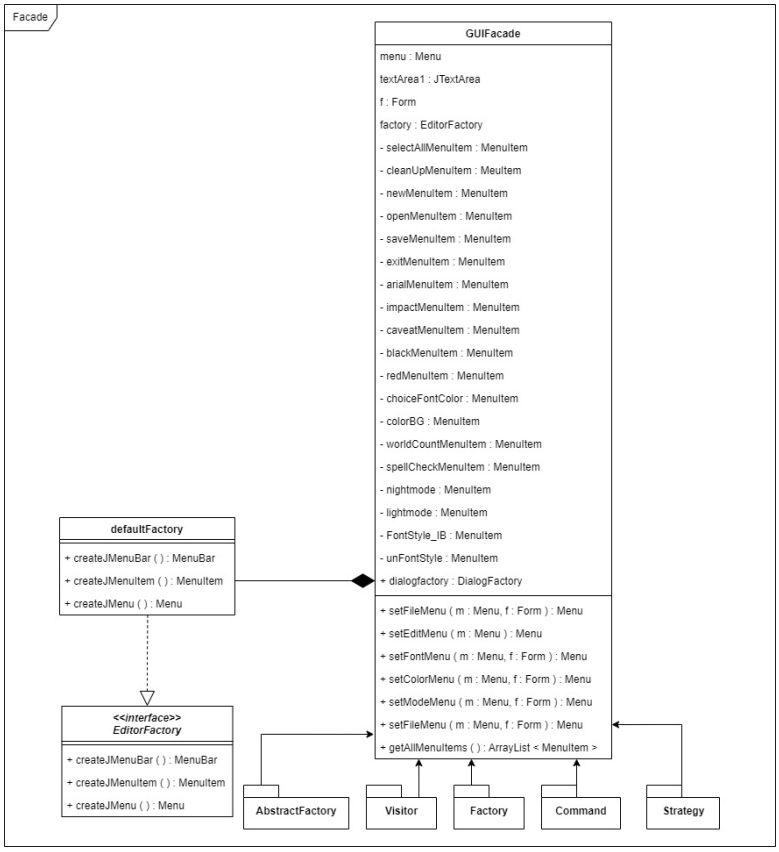
The number of class: 5

Inheritance: 2
Class bold is a subclass of class DecoratorStyle.
Class italic is a subclass of class DecoratorStyle.

Implement: 2
Class General implements class Componet.
Class DecoratorStyle implements class Componet.

The relationship of our design

Facade



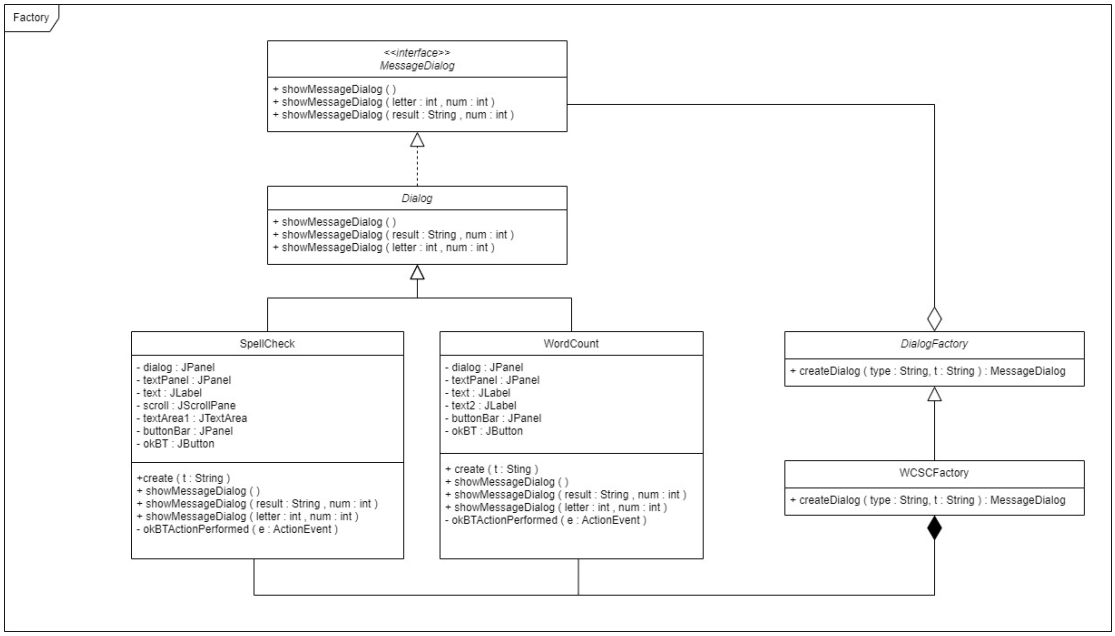
The number of class: 3

Composition: 1
Class defaultFactory is a part of class GUIFacade.

Implement: 1
Class defaultFactory implements class EditorFactory.

The relationship of our design

Factory



The number of class: 6

Aggregation: 1
Class DialogFactory owns class MessageDialog.

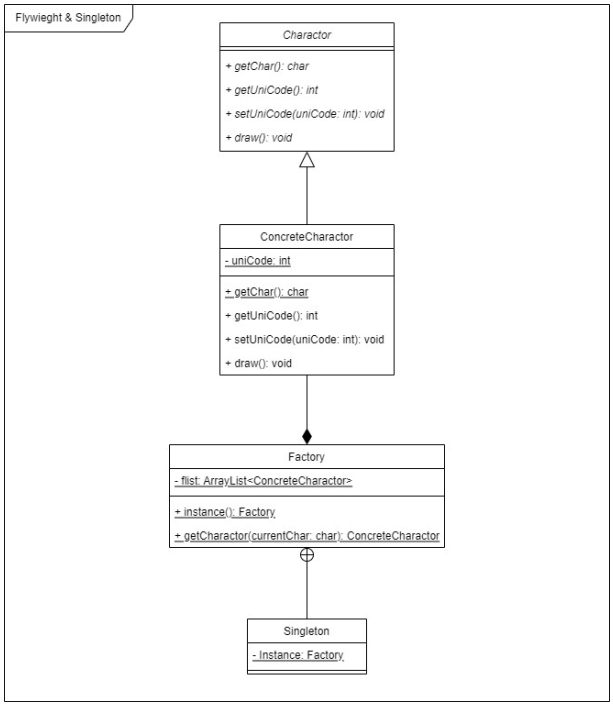
Composition: 2
Class SpellCheck is a part of class WCSCFactory.
Class WordCount is a part of class WCSCFactory.

Inheritance: 3
Class WCSCFactory is a subclass of class DialogFactory.
Class SpellCheck is a subclass of class Dialog.
Class WordCount is a subclass of class Dialog.

Implement: 1
Class Dialog implements class MessageDialog.

The relationship of our design

Flywieght & Singleton



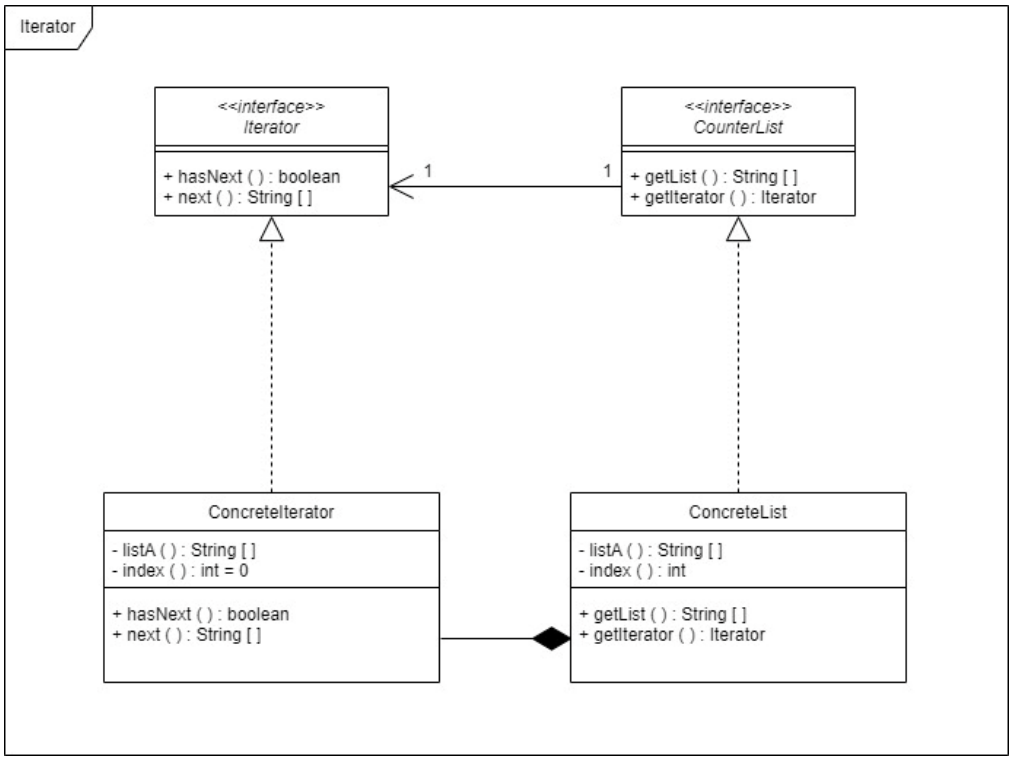
The number of class: 4

Composition: 1
Class ConcreteCharactor is a part of class Factory.

Inheritance: 1
Class ConcreteCharactor is a subclass of class Charactor.

The relationship of our design

Iterator



The number of class: 4

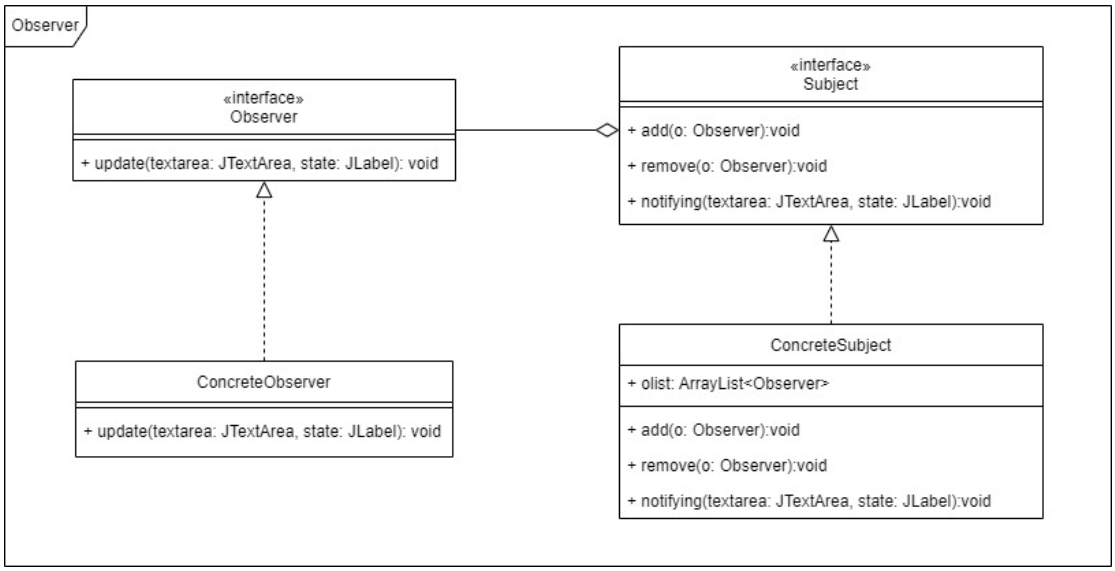
Association: 1
Class CounterList has class Iterator.

Composition: 1
Class ConcreteIterator is a part of class ConcreteList.

Implement: 2
Class ConcreteIterator implements class Iterator.
Class ConcreteList implements class CounterList.

The relationship of our design

Observer



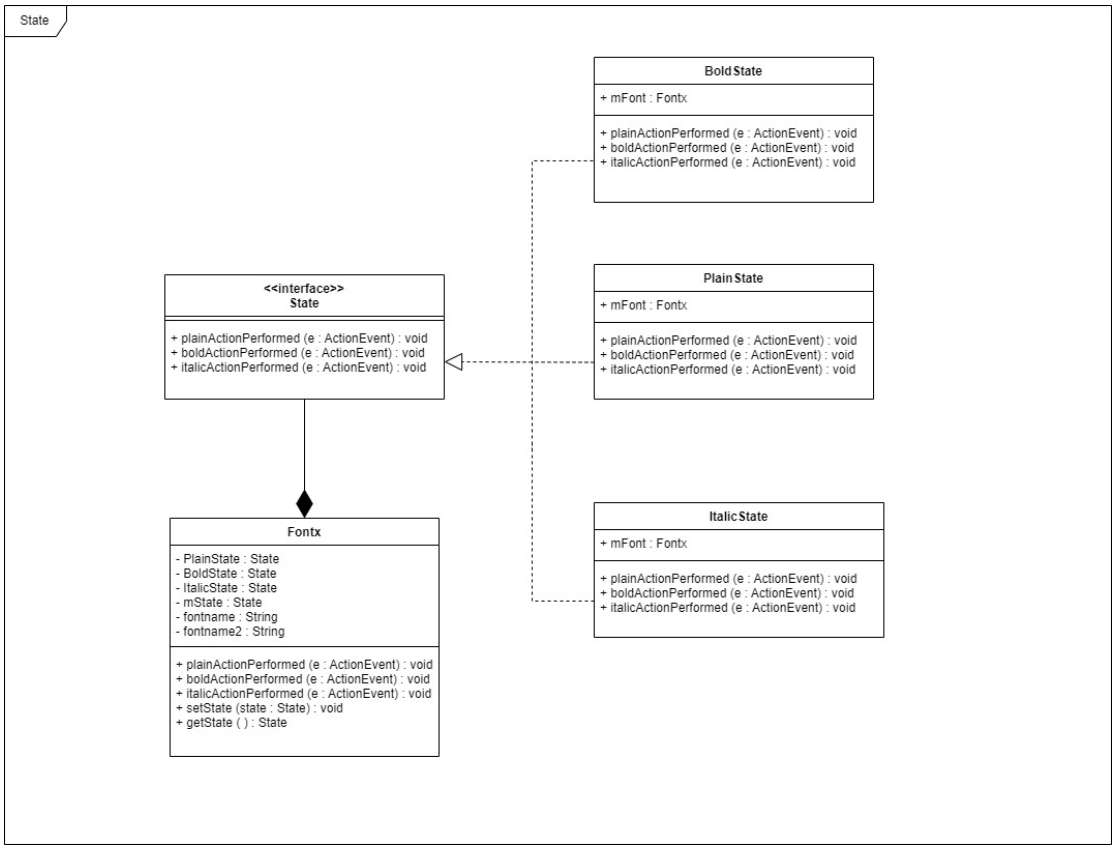
The number of class: 3

Aggregation: 1
Class Subject owns class Observer.

Implement: 2
Class ConcreteSubject implements class Subject.
Class ConcreteObserver implements class Observer.

The relationship of our design

State



The number of class: 5

Composition: 1

Class State is a part of class Fontx.

Implement: 3

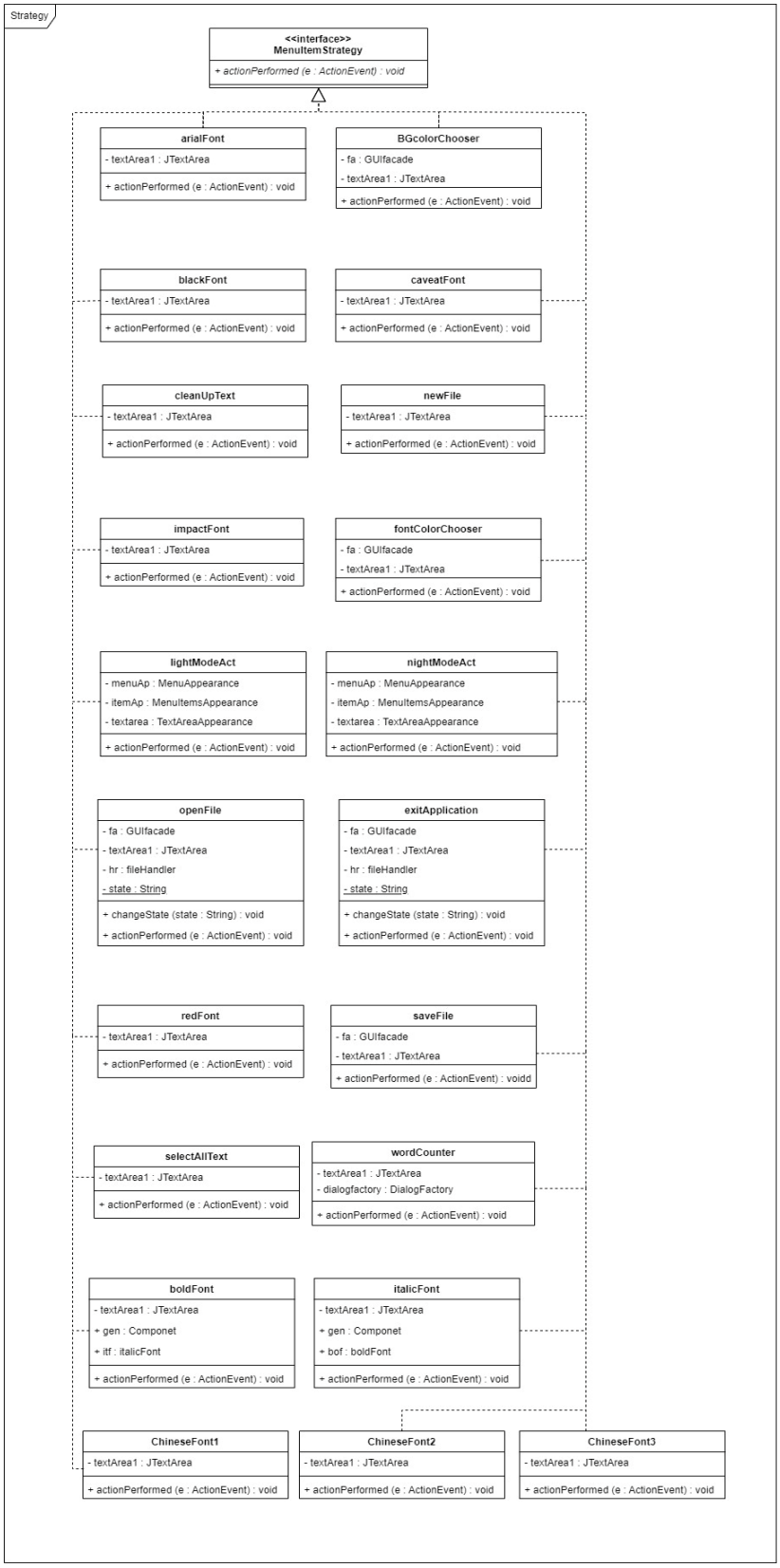
Class BoldState implements class State.

Class PlainState implements class State.

Class ltalicState implements class State.

The relationship of our design

Strategy



The relationship of our design

The number of class: 22

Implements: 21

Class BGcolorChooser implements class MenuItemStrategy.

Class ChineseFont1 implements class MenuItemStrategy.

Class ChineseFont2 implements class MenuItemStrategy.

Class ChineseFont3 implements class MenuItemStrategy.

Class arialFont implements class MenuItemStrategy.

Class blackFont implements class MenuItemStrategy.

Class boldFont implements class MenuItemStrategy.

Class caveatFont implements class MenuItemStrategy.

Class cleanUpText implements class MenuItemStrategy.

Class exitApplication implements class MenuItemStrategy.

Class fontColorChooser implements class MenuItemStrategy.

Class impactFont implements class MenuItemStrategy.

Class italicFont implements class MenuItemStrategy.

Class lightModeAct implements class MenuItemStrategy.

Class newFile implements class MenuItemStrategy.

Class nightModeAct implements class MenuItemStrategy.

Class openFile implements class MenuItemStrategy.

Class redFont implements class MenuItemStrategy.

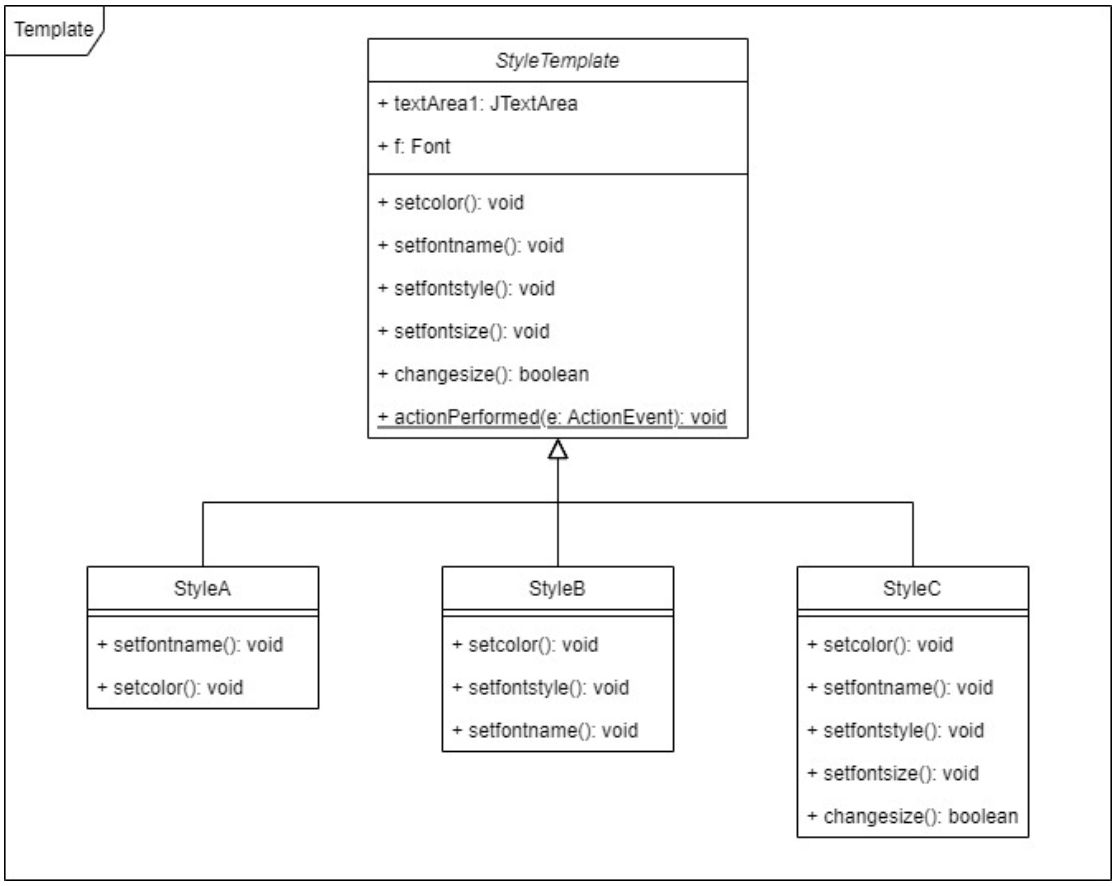
Class saveFile implements class MenuItemStrategy.

Class selectAllText implements class MenuItemStrategy.

Class wordCounter implements class MenuItemStrategy.

The relationship of our design

Template Method



The number of class: 4

Inheritance: 3

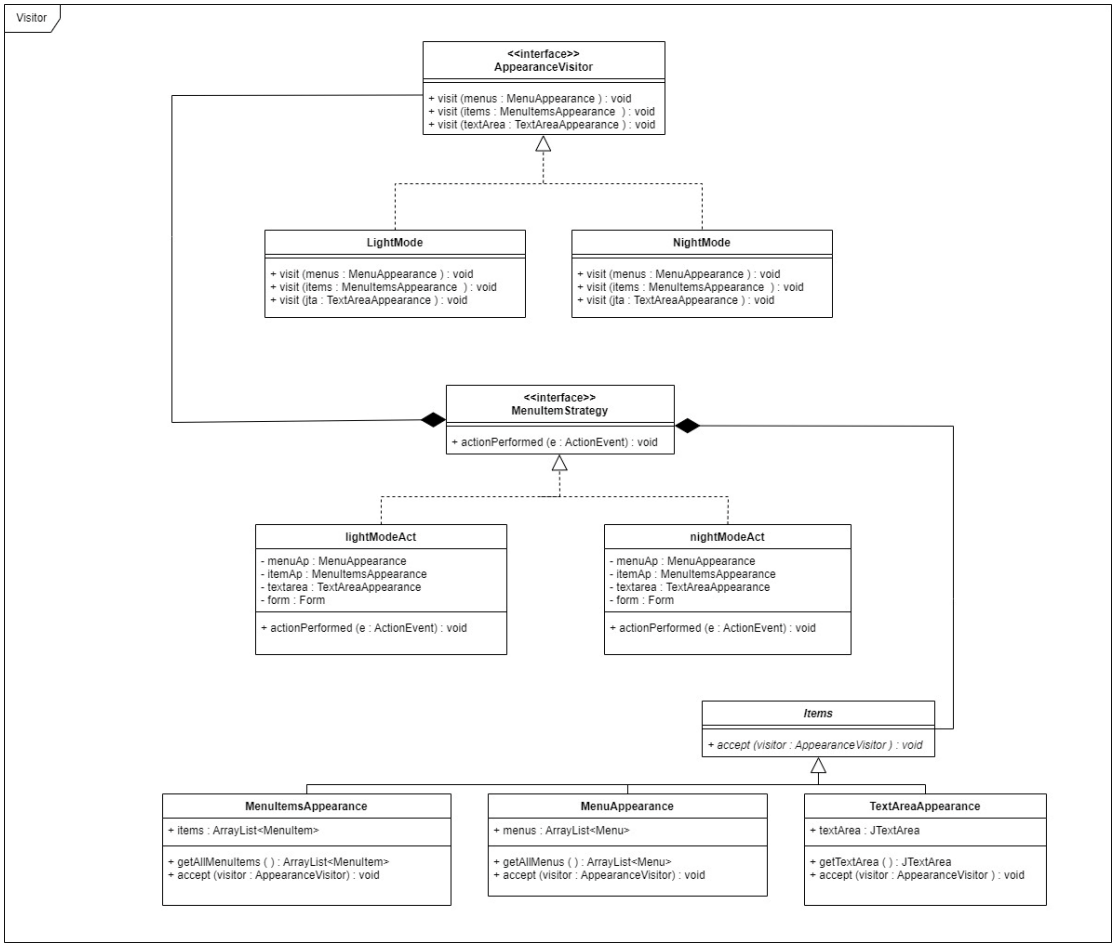
Class StyleA is a subclass of class StyleTemplate.

Class StyleB is a subclass of class StyleTemplate.

Class StyleC is a subclass of class StyleTemplate.

The relationship of our design

Visitor



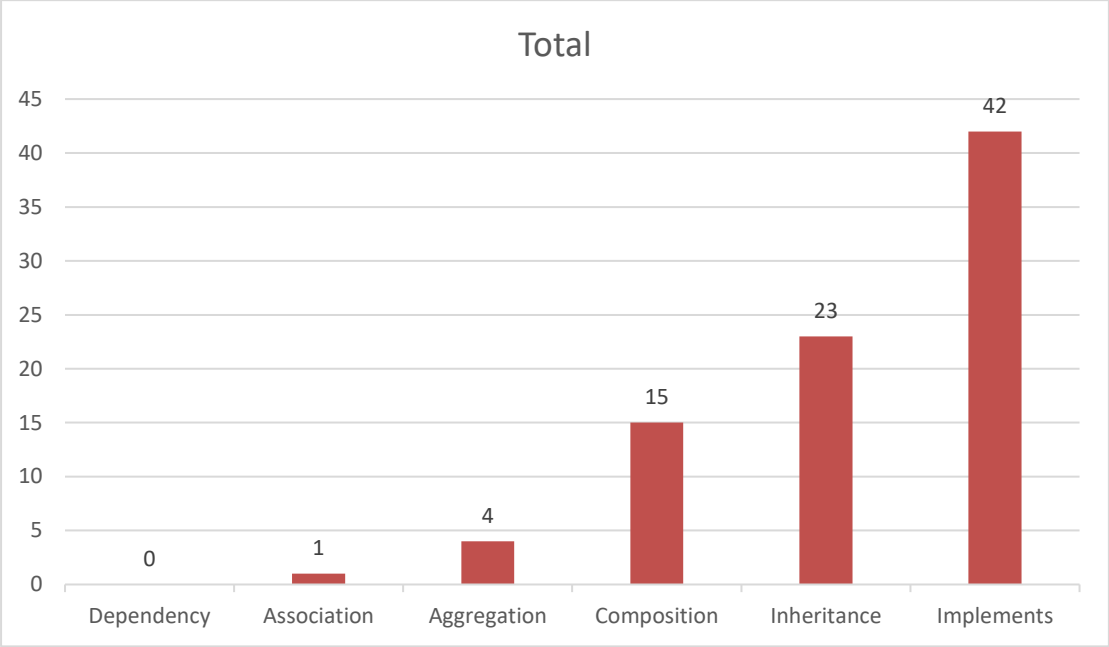
The number of class: 10

Composition: 2
Class Items is a part of class MenuItemStrategy.
Class AppearanceVisitor is a part of class MenuItemStrategy.

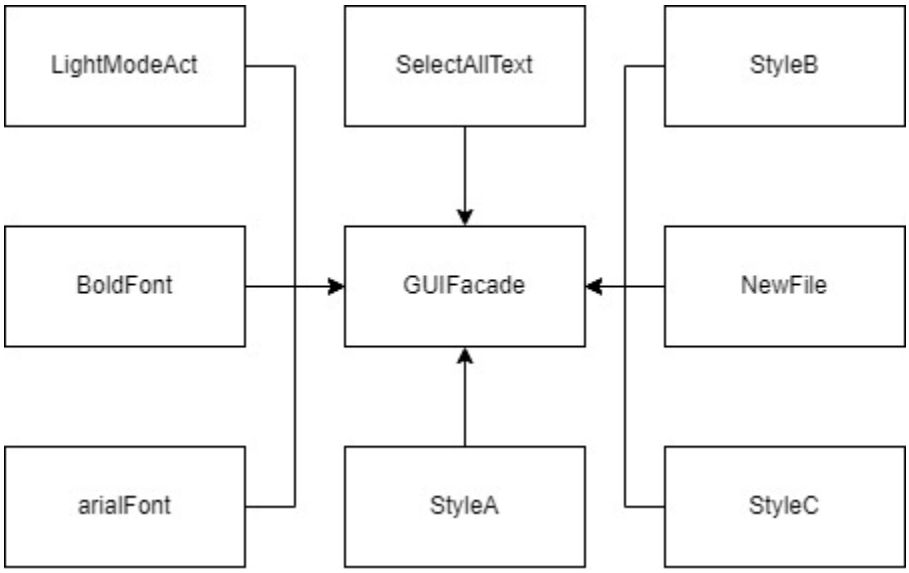
Inheritance: 3
Class TextAreaAppearance is a subclass of class Items.
Class MenuAppearance is a subclass of class Items.
Class MenuItemsAppearance is a subclass of class Items.

Implement: 4
Class LightMode implements class AppearanceVisitor.
Class NightMode implements class AppearanceVisitor.
Class lightModeAct implements class MenuItemStrategy.
Class nightModeAct implements class MenuItemStrategy.

Relationship in the whole system

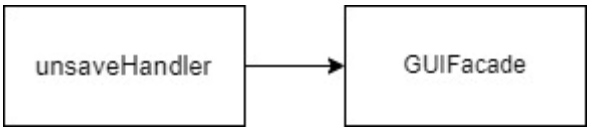


Fan-in



LightModeAct, BoldFont, arialFont, SelectAllText, NewFile, StyleA, StyleB, StyleC will ask for the data or things from GUTFacade test area.

Fan-out



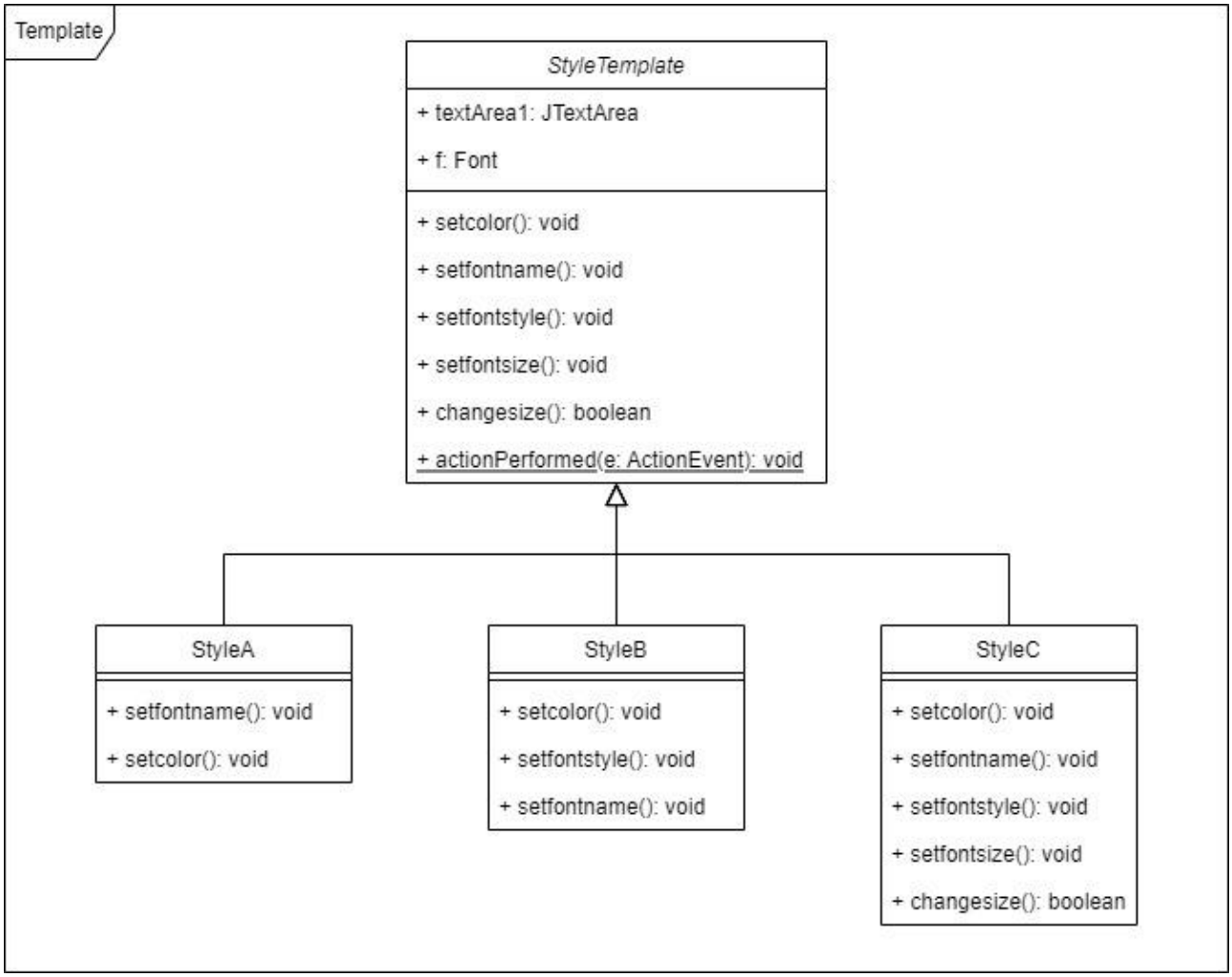
unsaveHandler will ask for GUIFacade.It will use GUIFacade and JtestArea and a variable State.

Three pieces of the design based on the change events

In the **Template Method pattern**, the class “StyleA”, “StyleB”, “StyleC” all extend the abstract class “StyleTemplate”.

That means the class “StyleTemplate” affects the class “StyleA”, “StyleB”, “StyleC”.

We choose the class “StyleC” for example. Because the class “StyleC” inherits the abstract class “StyleTemplate”. The method in the subclass will also change when the superclass changes the method.



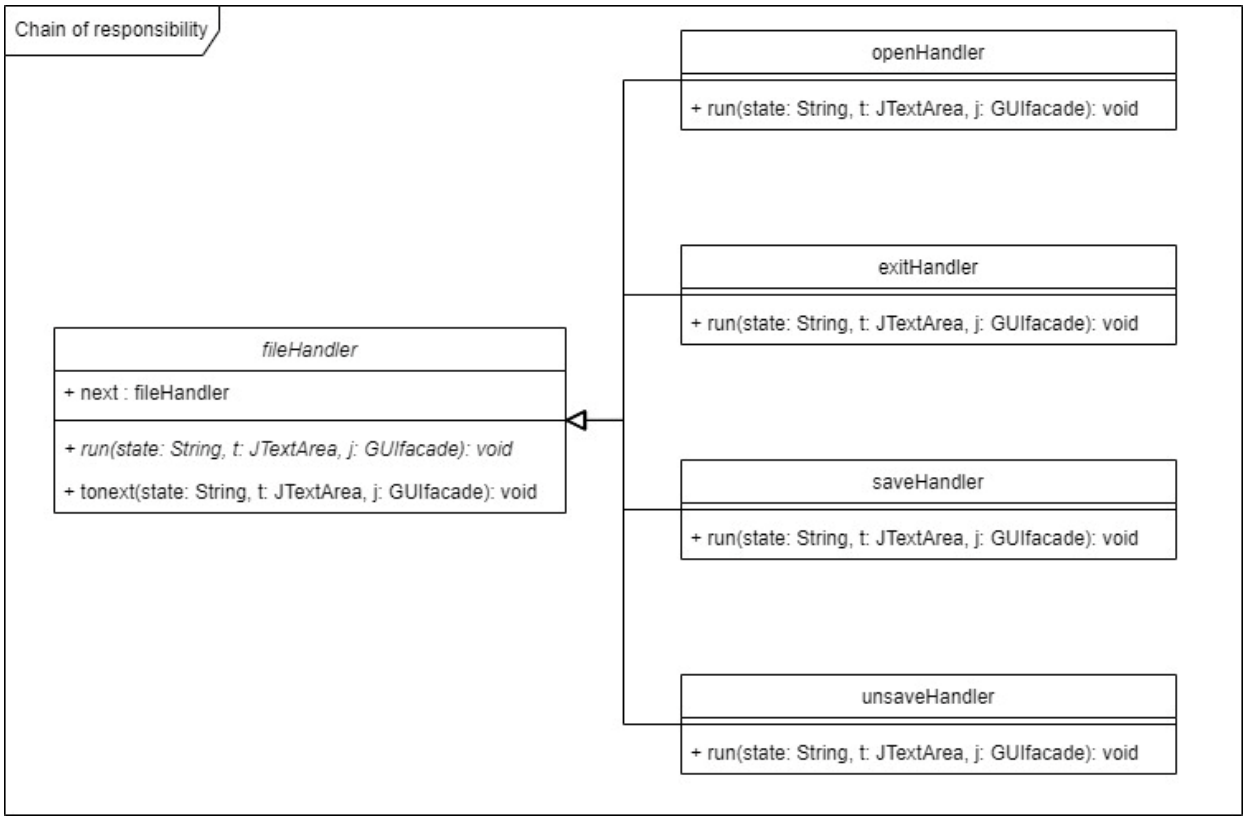
Three pieces of the design based on the change events

```java
abstract public class StyleTemplate implements ActionListener{
    JTextArea textArea1;
    Font f;
    public StyleTemplate(JTextArea t){
        textArea1 = t; //get JTextArea
    }
    public void setcolor() {
        textArea1.setForeground(new Color(0,0,0));
    }
    public void setfontname() {
        f = textArea1.getFont();
        textArea1.setFont(new Font("Arial",f.getStyle(),f.getSize()));
    }
    public void setfontstyle() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),Font.PLAIN,f.getSize()));
    }
    public void setfontsize() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),f.getStyle(),20));
    }
    public boolean changesize() {   //hookMethod
        return false;
    }

    final public void actionPerformed(ActionEvent e) { //TemplateMethod
        setcolor();
        setfontname();
        setfontstyle();
        if(changesize()) {
        setfontsize();
        }
     }
}
```

```java
public class StyleC extends StyleTemplate{
    public StyleC(JTextArea t){
        super(t);
    }
    @Override
    public void setcolor() {
        textArea1.setForeground(new Color(25,25,112));
    }
    @Override
    public void setfontname() {
        f = textArea1.getFont();
        textArea1.setFont(new Font("宋體",f.getStyle(),f.getSize()));
    }
    @Override
    public void setfontstyle() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),Font.BOLD+Font.ITALIC,f.getSize()));
    }
    @Override
    public void setfontsize() {
        f = textArea1.getFont();
        textArea1.setFont(new Font(f.getName(),f.getStyle(),36));
    }
    @Override
    public boolean changesize() {
        return true;
    }
}
```

Three pieces of the design based on the change events

In **Chain of Responsibility pattern**, the class “openHandler”, “exitHandler”, “saveHandler” and “unsaveHandler” all extend the abstract class “fileHandler”.
That means the abstract class “fileHandler” affects the class “openHandler”, “exitHandler”, “saveHandler” and “unsaveHandler”.
We choose the class “unsaveHandler” for example. Because the class “unsaveHandler” inherits the abstract class “fileHandler”. The method in the child class will also change when the super class changes the method.




```java
public abstract class fileHandler {
    //每個人都要知道下一個要處理的人是誰
    public fileHandler next=null;

    fileHandler(fileHandler next){
        this.next=next;
    }
    //傳入目前存檔的狀態用來判斷
    //傳入textarea跟GUIfacade是儲存跟開啟檔案的語法需要
    public abstract void run(String state,JTextArea t,GUIfacade j);
    //傳給下一個人
    public void tonext(String state,JTextArea t,GUIfacade j) {
        if(next!=null) {
            next.run(state,t,j);
        }else {
            j.closeDialog();
        }
    }
}
```

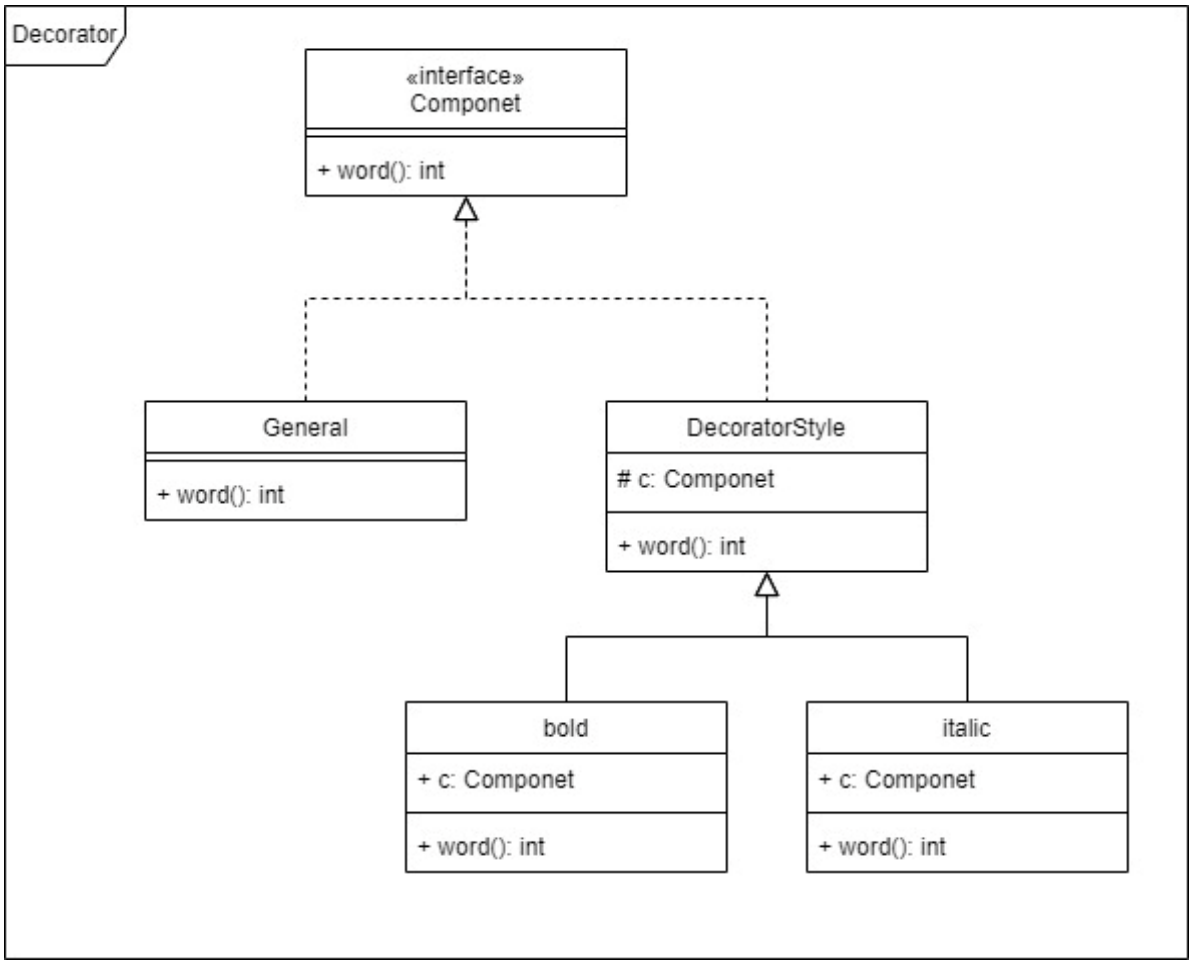
Three pieces of the design based on the change events

```
public class unsaveHandler extends fileHandler {
    public unsaveHandler(fileHandler next) {
        super(next);
    }
    public void run(String state,JTextArea t,GUIfacade j) {
        if(state=="no") {
            int option =JOptionPane.showConfirmDialog(null,
                    "檔案已修改，是否儲存？","儲存檔案？",
                    JOptionPane.YES_NO_OPTION,
                    JOptionPane.WARNING_MESSAGE,null);
            switch(option){
            // 確認檔案儲存
            case JOptionPane.YES_OPTION:state="save it";
            break;
            // 放棄檔案儲存
            case JOptionPane.NO_OPTION:
            state="done";
            break;
            }
        }
        tonext(state,t,j);
    }
}
```

Three pieces of the design based on the change events

In **Decorator pattern**, the class “General” and “DecoratorStyle” implements the interface “Componet”.That means the interface “Componet” affects the class “Componet” and “DecoratorStyle”.

We choose the class “DecoratorStyle” for example. Because the class “DecoratorStyle” must implement all methods in the interface “Componet”. The class “DecoratorStyle” will also change as long as the interface “Componet” changes the method.




```
public interface Componet {
    public int word();
}
```

```
public class DecoratorStyle implements Componet{
    protected Componet c;
    @Override
    public int word() {
        return 0;
    }
}
```

Teamwork Participation

Student ID	English name	Responsibility (Percentage) 負責項目/ 分工比例 ** 若同個項目有分工請記得寫 percentage	Score
B10821124	Leo	White/Black Box(including test code)(20%) Invocation chains(20%) Quality Metrics(20%) Explain the relationships of our system(20%) Describe change events(20%)Class diagram(20%)	100%
B10823011	Kousa	System code (20%) Pattern code(20%) Debug(20%) Integration of code(20%) Revised TW1 code(20%)	100%
B10823015	Debby	System code (20%) Pattern code(20%) Debug(20%) Integration of code(20%) Revised TW1 code(20%) Explain Pattern(16.5%)	100%
B10823016	Kris	System code (20%) Pattern code(20%) Debug(20%) Integration of code(20%) Revised TW1 code(20%) Explain Pattern(33%)	100%
B10823018	Bob	System code (20%) Pattern code(20%) Debug(20%) Integration of code(20%) Revised TW1 code(20%) Explain Pattern(16.5%)	100%
B10823024	Michael	White/Black Box(including test code)(20%) Invocation chains(20%) Quality Metrics(20%) Explain the relationships of our system(20%) Describe change events(20%)Class diagram(20%)	100%
B10823029	Leo	System code (20%) Pattern code(20%) Debug(20%) Integration of code(20%) Revised TW1 code(20%)	100%
B10823031	Andrew	White/Black Box(including test code)(20%) Invocation chains(20%) Quality Metrics(20%) Explain the relationships of our system(20%) Describe change events(20%)Class diagram(20%)	100%
B10823038	Joanne	White/Black Box(including test code)(20%) Invocation chains(20%) Quality Metrics(20%) Explain the relationships of our system(20%) Describe change events(20%)Class diagram(20%) Explain Pattern(33%)	100%

Student ID	English name	Responsibility (Percentage) 負責項目/ 分工比例 ** 若同個項目有分工請記得寫 percentage	Score
B11023069	Young	White/Black Box(including test code)(20%) Invocation chains(20%) Quality Metrics(20%) Explain the relationships of our system(20%) Describe change events(20%)Class diagram(20%)	100%