设计模式实验 17-20

一、实验目的

1.结合实例，熟练绘制设计模式结构图。
2.结合实例，熟练使用 Java 语言实现设计模式。
3.通过本实验，理解每一种设计模式的模式动机，掌握模式结构，学习如何使用代码实现这些设计模式。

二、实验要求

1.结合实例，绘制设计模式的结构图。
2.使用 Java 语言实现设计模式实例，代码运行正确。

三、实验内容

1. 观察者模式：某文字编辑软件须提供如下功能：在文本编辑窗口中包含一个可编辑文本区和 3 个文本 信息统计区，用户可以在可编辑文本区对文本进行编辑操作，第一个文本信息统计区用于显 示可编辑文本区中出现的单词总数量和字符总数量，第二个文本信息统计区用于显示可编辑 文本区中出现的单词（去重后按照字典序排序），第三个文本信息统计区用于按照出现频次 降序显示可编辑文本区中出现的单词以及每个单词出现的次数（例如 hello :5）。现采用观 察者模式设计该功能，绘制对应的类图并编程模拟实现。

2. 备忘录模式：某文字编辑软件须提供撤销（Undo）和重做 / 恢复（Redo）功能，并且该软件可支持 文档对象的多步撤销和重做。开发人员决定采用备忘录模式来实现该功能，在实现过程中引 人栈（Stack）作为数据结构。在实现时，可以将备忘录对象保存在两个栈中，一个栈包含 用于实现撤销操作的状态对象，另一个栈包含用于实现重做操作的状态对象。在实现撤销操 作时，会弹出撤销栈栈顶对象以获取前一个状态并将其设置给应用程序；同样，在实现重做 操作时，会弹出重做栈栈顶对象以获取下一个状态并将其设置给应用程序。绘制对应的类图 并编程模拟实现。

3. 状态模式：在某网络管理软件中， TCP 连接（TCP Connection）具有建立（Established）、监听 （Listening）、关闭（Closed）等多种状态，在不同的状

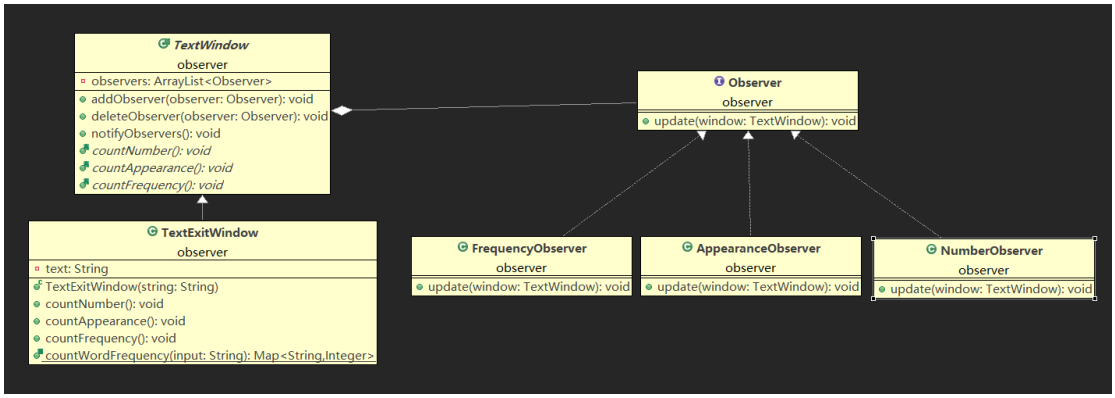
态下 TCP 连接对象具有不同的行 为，连接对象还可以从一个状态转换到另一个状态。当一个连接对象收到其他对象的请求时， 它根据自身的当前状态做出不同的反应。现采用状态模式对 TCP 连接进行设计，绘制对应 的类图并编程模拟实现。

4. 享元模式：某 OA 系统采用享元模式设计权限控制与管理模块，在该模块中，将与系统功能相对 应的业务类设计为享元类并将相应的业务对象存储到享元池中（提示；可使用 Map 实现， key 为业务对象对应的权限编码， value 为业务对象）。用户身份验证成功后，系统通过 存储在数据库中的该用户的权限编码集从享元池获取相应的业务对象并构建权限列表，在界 面上显示用户所拥有的权限。根据以上描述，绘制对应的类图并编程模拟实现。

四、实验结果

需要提供设计模式实例的结构图（类图）和实现代码。

1. 观察者模式



代码实现：

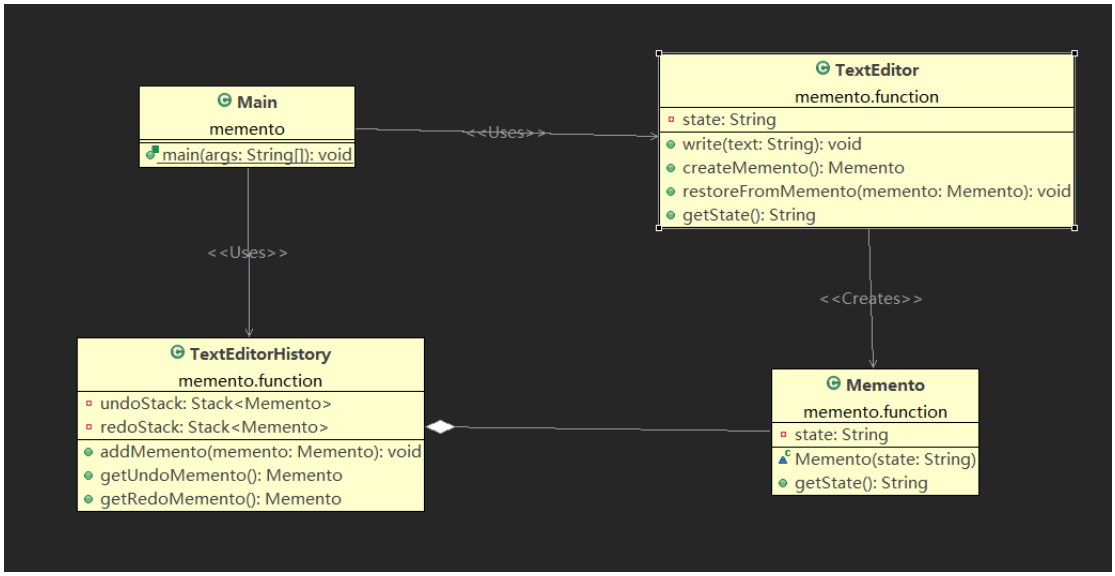
```
public interface Observer {

    public void update(TextWindow window);

}

public class NumberObserver implements Observer{

    @Override
    public void update(TextWindow window) {
        window.countNumber();
        System.out.println("");

    }
```

```
}

public class FrequencyObserver implements Observer{

    @Override
    public void update(TextWindow window) {
        window.countFrequency();
        System.out.println("");

    }

}

public class AppearanceObserver implements Observer{

    @Override
    public void update(TextWindow window) {
        window.countAppearance();
        System.out.println("");

    }

}

public abstract class TextWindow {

    private ArrayList<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void deleteObserver(Observer observer) {
        observers.remove(observer);
    }

    public void notifyObservers() {
        Iterator it = observers.iterator();
        while(it.hasNext()) {
            Observer observer = (Observer)it.next();
            observer.update(this);
        }
    }

    public abstract void countNumber();
    public abstract void countAppearance();
    public abstract void countFrequency();
}

public class TextExitWindow extends TextWindow{

    private String text;

    public TextExitWindow(String string) {
        this.text = string;
    }
```

```
    @Override
    public void countNumber() {
         String[] words = text.split("\\s+");
        String stringWithoutSpaces = text.replaceAll("\\s", "");
        System.out.println("单词总数: " + words.length);
        System.out.println("字符总数: " + stringWithoutSpaces.length());

    }

    @Override
    public void countAppearance() {
         String[] words = text.split("\\s+");
         TreeSet<String> uniqueWordSet = new TreeSet<>(Arrays.asList(words));
         String[] uniqueWords = uniqueWordSet.toArray(new String[0]);
         Arrays.sort(uniqueWords);
        System.out.println("去重后按字典序排序的单词列表:");
        for (String word : uniqueWords) {
            System.out.println(word);
        }

    }

    @Override
    public void countFrequency() {
         Map<String, Integer> wordFrequencyMap = countWordFrequency(text);

         List<Entry<String, Integer>> wordFrequencyList = new
ArrayList<>(wordFrequencyMap.entrySet());

        Collections.sort(wordFrequencyList, new Comparator<Entry<String, Integer>>()
{
            @Override
            public int compare(Entry<String, Integer> entry1, Entry<String, Integer>
entry2) {
                return entry2.getValue().compareTo(entry1.getValue());
            }
        });

        for (Entry<String, Integer> entry : wordFrequencyList) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }


    }

    public static Map<String, Integer> countWordFrequency(String input) {
        String[] words = input.split("\\s+");
        Map<String, Integer> wordFrequencyMap = new HashMap<>();

        for (String word : words) {
            word = word.replaceAll("[^a-zA-Z]", "").toLowerCase();
            wordFrequencyMap.put(word, wordFrequencyMap.getOrDefault(word, 0) + 1);
        }

        return wordFrequencyMap;
    }

}
```

```
public class Main {

    public static void main(String[] args) {
        TextWindow window = new TextExitWindow("this is a text window of text");
        window.addObserver(new NumberObserver());
        window.addObserver(new AppearanceObserver());
        window.addObserver(new FrequencyObserver());
        window.notifyObservers();
    }

}
```

2. 备忘录模式



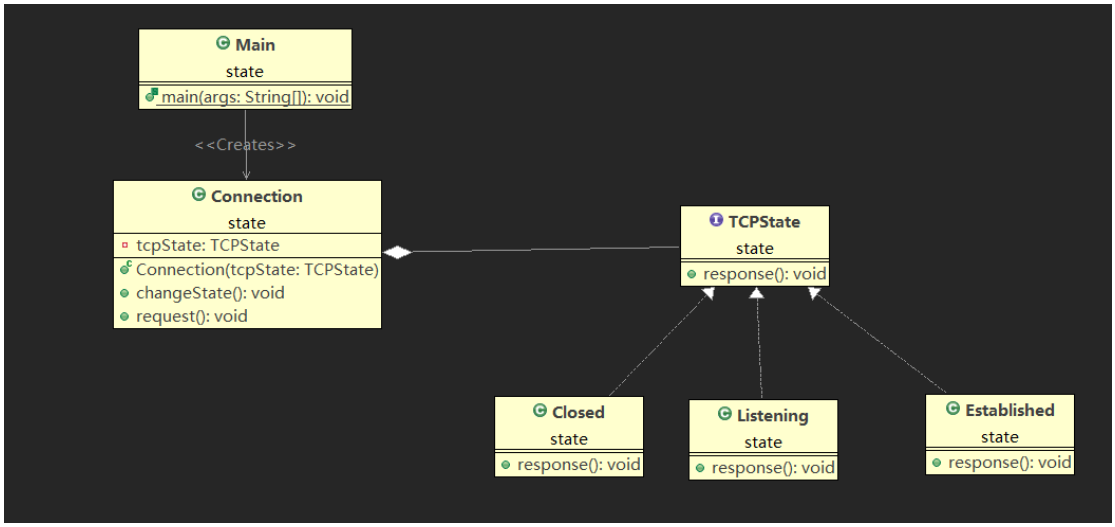
代码实现：

```
public class Memento {

    private String state;

    Memento(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }

}

public class TextEditor {

    private String state = "";

    public void write(String text) {
        state += text;
    }
```

```
    public Memento createMemento() {
        return new Memento(this.state);
    }

    public void restoreFromMemento(Memento memento) {
        state = memento.getState();
    }

    public String getState() {
        return state;
    }

}

public class TextEditorHistory {

     private Stack<Memento> undoStack = new Stack<>();
    private Stack<Memento> redoStack = new Stack<>();

    public void addMemento(Memento memento) {
        undoStack.push(memento);
        redoStack.clear();
    }

    public Memento getUndoMemento() {
        if (!undoStack.isEmpty()) {
            Memento memento = undoStack.pop();
            Memento memento1 = undoStack.pop();
            redoStack.push(memento);
            return memento1;
        }
        return null;
    }

    public Memento getRedoMemento() {
        if (!redoStack.isEmpty()) {
            Memento memento = redoStack.pop();
            undoStack.push(memento);
            return memento;
        }
        return null;
    }
}

public class Main {

     public static void main(String[] args) {
         TextEditor editor = new TextEditor();
        TextEditorHistory history = new TextEditorHistory();

        editor.write("Hello ");
        history.addMemento(editor.createMemento());

        editor.write("world ");
        history.addMemento(editor.createMemento());

        System.out.println("Current Content: " + editor.getState());
```

```
        Memento undoMemento = history.getUndoMemento();
        if (undoMemento != null) {
            editor.restoreFromMemento(undoMemento);
            System.out.println("After Undo: " + editor.getState());
        }

        Memento redoMemento = history.getRedoMemento();
        if (redoMemento != null) {
            editor.restoreFromMemento(redoMemento);
            System.out.println("After Redo: " + editor.getState());
        }
    }

}
```

3. 状态模式



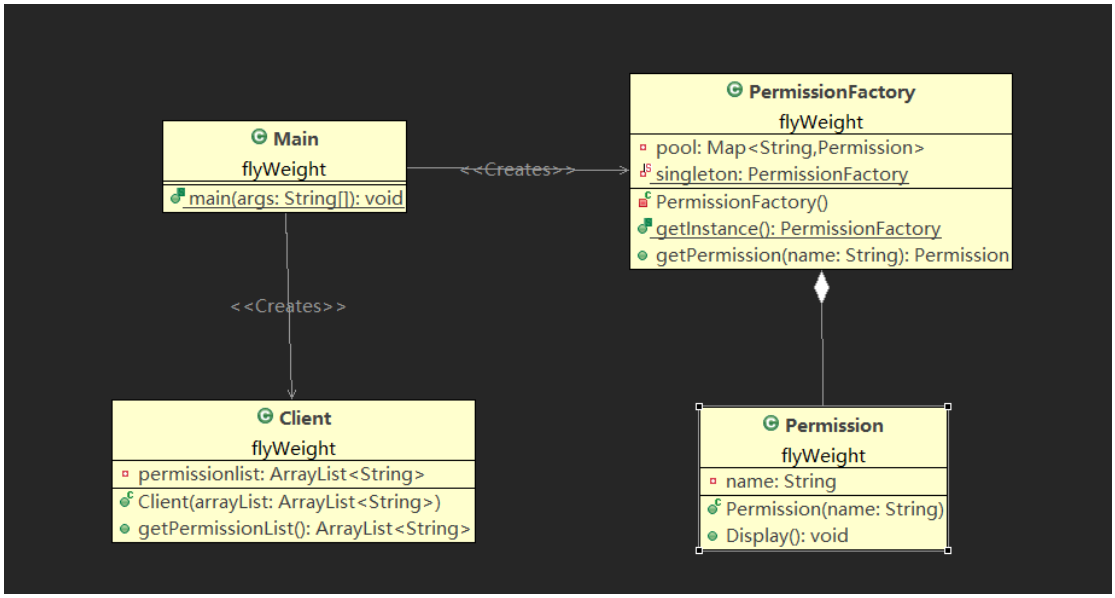
代码实现：

```
public interface TCPState {

    public void response();
    public int getSymbol();

}

public class Established implements TCPState{

    private int symbol = 0;

    @Override
    public void response() {
        System.out.println("Do Established TCP Response");

    }

    @Override
    public int getSymbol() {
```

```
        return symbol;
    }

}

public class Listening implements TCPState{

    private int symbol = 1;

    @Override
    public void response() {
        System.out.println("Do Listening TCP Response");

    }

    @Override
    public int getSymbol() {
        return symbol;
    }

}

public class Closed implements TCPState{

    private int symbol = 2;

    @Override
    public void response() {
        System.out.println("Do Closed TCP Response");

    }

    @Override
    public int getSymbol() {
        return symbol;
    }

}

public class Connection {

    private TCPState tcpState;

    public Connection(TCPState tcpState) {
        this.tcpState = tcpState;
    }

    public void changeState() {
        if(tcpState.getSymbol() == 0) {
            tcpState = new Listening();
        }
        else if(tcpState.getSymbol() == 1) {
            tcpState = new Closed();
        }
        else if(tcpState.getSymbol() == 2) {
            tcpState = new Established();
        }
    }
```

```
    public void request() {
        tcpState.response();
    }

}

public class Main {

    public static void main(String[] args) {
        TCPState tcpState = new Established();
        Connection connection = new Connection(tcpState);
        connection.request();
        connection.changeState();
        connection.request();
        connection.changeState();
        connection.request();
    }

}
```

4. 享元模式



代码实现：

```
public class Permission {

    private String name;

    public Permission(String name) {
        this.name = name;
        System.out.println("Creating Permission ***" + name + "***");
    }

    public void Display() {
        System.out.println("The Permission is " + name);
    }
```

```
}

public class PermissionFactory {

    private Map<String, Permission> pool = new HashMap<>();
    private static PermissionFactory singleton = new PermissionFactory();

    private PermissionFactory() {}

    public static PermissionFactory getInstance() {
        return singleton;
    }

    public synchronized Permission getPermission(String name) {
        Permission permission = (Permission)pool.get(name);
        if(permission == null) {
            permission = new Permission(name);
            pool.put(name, permission);
        }
        return permission;
    }


}

public class Client {

    private ArrayList<String> permissionlist;

    public Client(ArrayList<String> arrayList) {
        permissionlist = arrayList;
    }

    public ArrayList<String> getPermissionList(){
        return permissionlist;
    }

}

public class Main {

    public static void main(String[] args) {
        PermissionFactory pFactory;
        pFactory = PermissionFactory.getInstance();

        ArrayList<String> c1 = new ArrayList<>();
        ArrayList<String> c2 = new ArrayList<>();

        c1.add("read");
        c2.add("read");
        c2.add("write");

        Client cl1 = new Client(c1);
        Client cl2 = new Client(c2);

        ArrayList<String> p1 = cl1.getPermissionList();
        ArrayList<String> p2 = cl2.getPermissionList();
```

```
        for(String p: p1) {
            Permission permission = pFactory.getPermission(p);
            permission.Display();
        }
        for(String p: p2) {
            Permission permission = pFactory.getPermission(p);
            permission.Display();
        }
    }

}
```

五、实验小结

请总结本次实验的体会，包括学会了什么、遇到哪些问题、如何解决这些问题

以及存在哪些有待改进的地方。

通过实际编写代码，我加强了对观察者模式、备忘录模式、状态模式，享元模式四种设计模式的认识，了解了这四种设计模式在实际运用中的作用和意义。

备忘录模式中，遇到的问题：一次 undo 之后，没有实现撤销效果。
解决办法：当完成一次操作后，就将其状态压进 undo 栈中，再希望撤销时弹栈，就为当前状态，故需要弹栈两次才为上一个状态，而第一次弹栈的状态压栈进入 redo 栈。