设计模式实验（4）

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. 组合模式

某移动社交软件要增加一个群组（Group）功能。通过设置，用户可以将自己的动态信息（包括最新动态、新上传的视频以及分享的链接等）分享给某个特定的成员（Member）.也可以分享给某个群组中的所有成员；用户可以将成员加至某个指定的群组；此外，还允许用户在一个群组中加子群组，以便更加灵活地实现面向特定人群的信息共享。现采用组合模式设计该群组功能，绘制对应的类图并编程模拟实现。

2. 装饰模式

在某 OA 系统中提供一个报表生成工具，用户可以通过该工具为报表增加表头和表尾，允许用户为报表增加多个不同的表头和表尾，用户还可以自行确定表头和表尾的次序。为了能够灵活设置表头和表尾的次序并易于增加新的表头和表尾，现采用装饰模式设计该报表生成工具，绘制对应的类图并编程模拟实现。

3. 访问者模式

某软件公司需要设计一个源代码解析工具，该工具可以对源代码进行解析和处理，在该工具的初始版本中，主要提供了以下3 个功能。

- (1)度量软件规模。可以统计源代码中类的个数、每个类属性的个数以及每个类方法的个数等。
- (2)提取标识符名称，以便检查命名是否合法和规范。可以提取类名、属性名和方法名等。
- (3)统计代码行数。可以统计源代码中每个类和每个方法中源代码的行数。
  将来还会在工具中增加一些新功能，为源代码中的类、属性和方法等提供更多的解析操作。现采用访问者模式设计该源代码解析工具，可将源代码中的类、属性和方法等设计为待访问的元素，上述不同功能由不同的具体访问者类实现，绘制对应的类图并编程模拟实现。

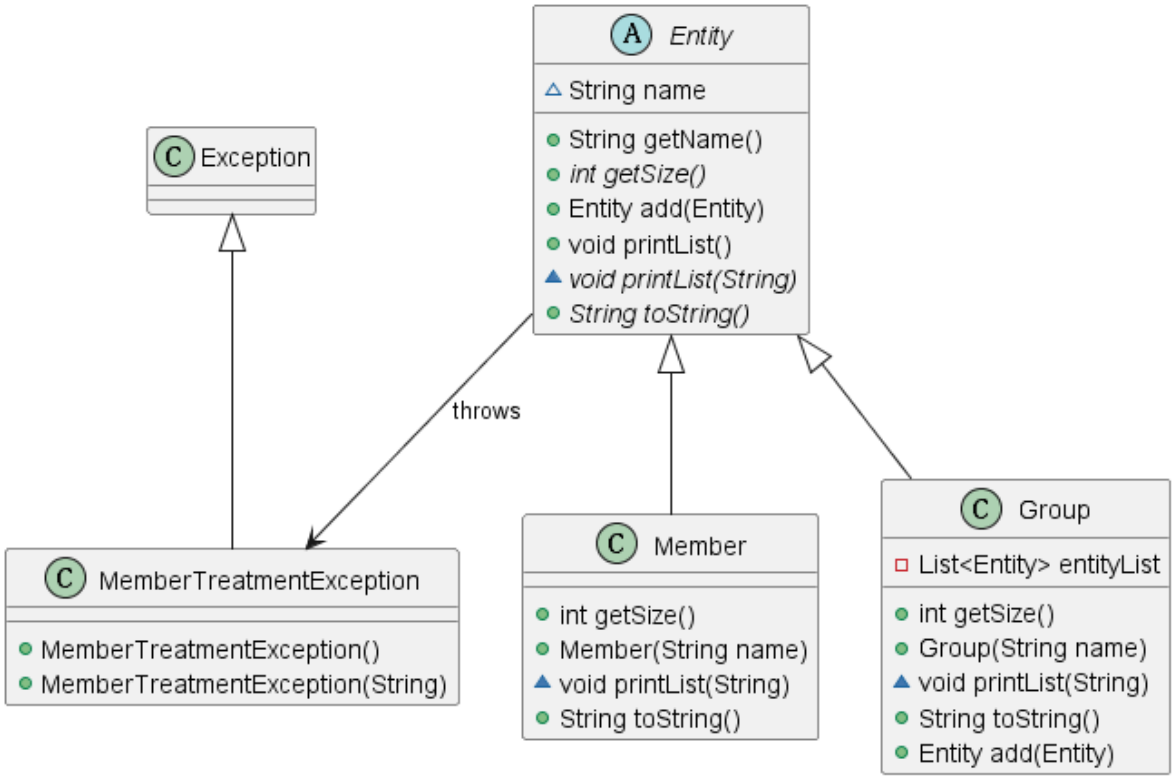
4. 职责链模式

在某 Web 框架中采用职责链模式来组织数据过滤器，不同的数据过滤器提供了不同的功能，例如字符编码转换过滤器、数据类型转换过滤器、数据校验过滤等，可以将多个过滤器连接成一一个过滤器链，进而对数据进行多次处理。根据以上描述，绘制对应的类图并编程模拟实现。

三、实验结果

1.组合模式

类图



实现代码

- Entity.java

```
public abstract class Entity {
    protected String name;

    public Entity(String name){
        this.name = name;
    }

    public String getName(){
        return this.name;
    };
    public abstract int getSize();

    public Entity add(Entity entity) throws MemberTreatmentException{
        throw new MemberTreatmentException();
    }
    public void  printList(){
        printList("");
    }

    protected abstract void printList(String prefix);

    public abstract String toString();
}
```

- Group.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Group extends Entity{

    private List<Entity> entityList = new ArrayList<Entity>();

    public Group(String name) {
        super(name);
    }

    @Override
    public int getSize() {
        int sz = 0;
        Iterator it = entityList.iterator();
        while(it.hasNext()){
            sz ++;
            it.next();
        }
        return sz;
    }

    @Override
    protected void printList(String prefix) {
        System.out.println(prefix + this.toString());
        Iterator it = entityList.iterator();
        while(it.hasNext()){
            ((Entity)it.next()).printList(prefix+"---  ");
        }
    }

    public Entity add(Entity entity){
        entityList.add(entity);
        return this;
    }

    @Override
    public String toString() {
        return "{ " + this.name + " , " + this.getSize() + " } ";
    }
}
```

- Member.java

```
public class Member extends Entity{

    public Member(String name){
        super(name);
    }

    @Override
    public int getSize() {
        return 1;
    }

    @Override
```

```
    protected void printList(String prefix) {
        System.out.println(prefix + this.toString());
    }

    @Override
    public String toString() {
        return "[ " + this.name + " ]";
    }
}
```

- MemberTreatmentException.java

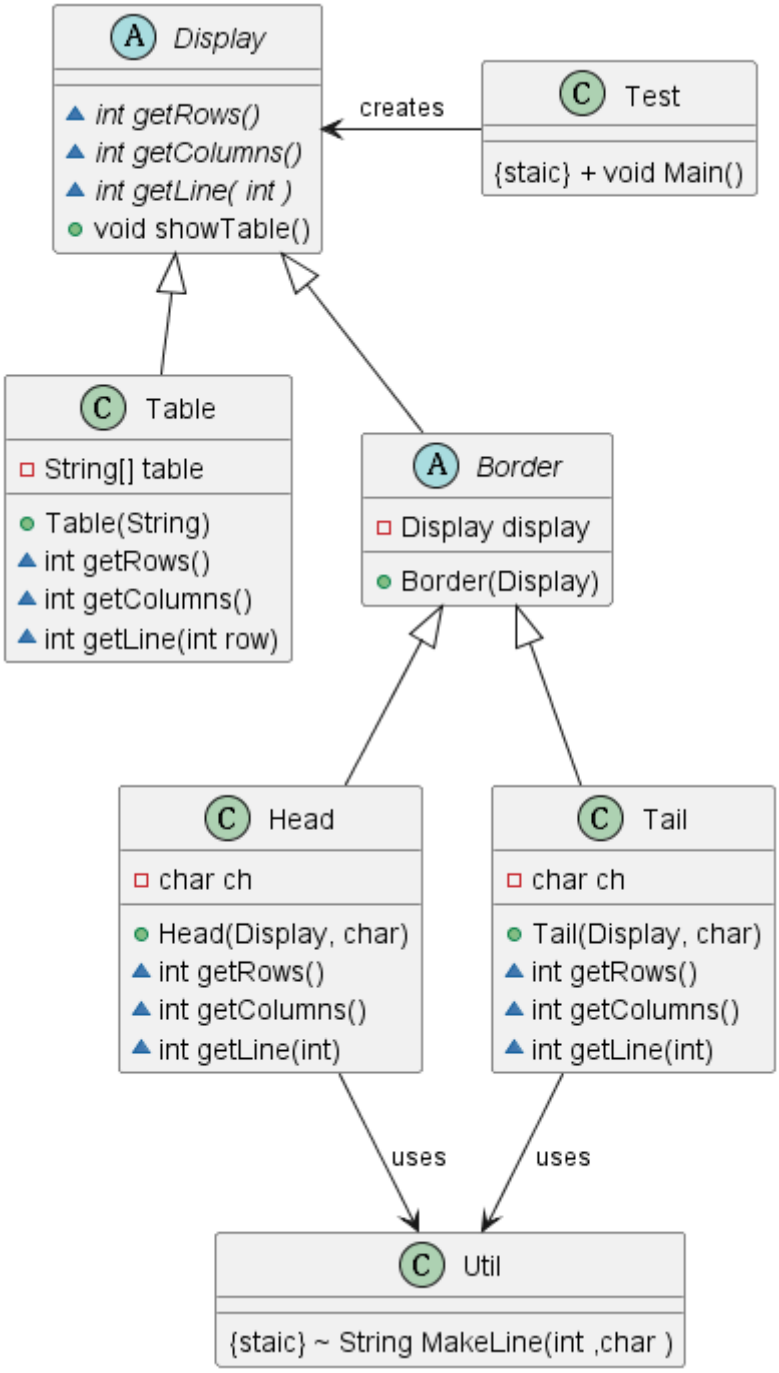
```
public class MemberTreatmentException extends RuntimeException{
    public MemberTreatmentException(){}
    public MemberTreatmentException(String msg){
        super(msg);
    }
}
```

- Test_4_1.java

```
public class Test_4_1 {
    public static void main(String[] args) {
        System.out.println();
        Group groupA = new Group("Group A");
        Group groupB = new Group("Group B");
        Member memberA = new Member("Member A");
        Member memberB = new Member("Member B");
        groupA.add(memberA);
        groupA.add(memberB);
        groupB.add(memberB);
        groupB.add(groupA);
        Group home = new Group("Home");
        home.add(groupB);
        Group groupC = new Group("Group C");
        Group groupD = new Group("Gropu D");
        Group groupE = new Group("Group E");
        Member memberE = new Member("Member E");
        groupE.add(memberE);
        home.add(groupD);
        home.add(groupE);
        home.add(groupC);
        home.printList();
    }
}
```

2. 装饰模式

类图



实现代码

- Border.java

```
public abstract class Border extends Display {
    protected Display display;
    public Border(Display display){
        this.display=display;
    }
}
```

- Display.java

```
public abstract class Display {
    abstract int getColumns();
    abstract int getRows();
    abstract String getLine(int row);
    public void showTable(){
        for (int i=0;i<getRows();i++){
            System.out.println(getLine(i));
        }
        System.out.println("\n");
    }
}
```

- Head.java

```
public class Head extends Border{

    private char ch;

    public Head(Display display , char ch) {
        super(display);
        this.ch = ch;
    }

    @Override
    int getColumns() {
        return display.getColumns();
    }

    @Override
    int getRows() {
        return display.getRows()+3;
    }

    @Override
    String getLine(int row) {
        if(row==0||row==2){
            return Util.MakeLine(display.getColumns(),ch);
        } else if(row == 1){
            return Util.MakeLine((display.getColumns()-6)/2 ,ch) +
                    " Head " +
                    Util.MakeLine((display.getColumns()-5)/2 ,ch);
        } else return display.getLine(row-3);
    }
}
```

- Table.java

```
public class Table extends Display {
    String []table  ;

    public Table(String content){
        table = new String[3];
        table[1]="="+content+"=";
        table[2]=table[0]=new String(Util.MakeLine(content.length()+2,'='));
    }
```

```
    @Override
    int getColumns() {
        return table[1].length();
    }

    @Override
    int getRows() {
        return 3;
    }

    @Override
    String getLine(int row) {
        if(row<3)return table[row];
        else return null;
    }
}
```

- Tail.java

```
public class Tail extends Border{

    private char ch;

    public Tail(Display display,char ch) {
        super(display);
        this.ch = ch;
    }

    @Override
    int getColumns() {
        return display.getColumns();
    }

    @Override
    int getRows() {
        return display.getRows()+3;
    }

    @Override
    String getLine(int row) {
        if(row < display.getRows()){
            return display.getLine(row);
        } else {
            row -= display.getRows();
            if(row==0||row==2){
                return Util.MakeLine(display.getColumns(),ch);
            } else return Util.MakeLine((display.getColumns()-6)/2 ,ch) +
                        " Tail " +
                        Util.MakeLine((display.getColumns()-5)/2 ,ch);
        }
    }
}
```

- Test_4_2.java

```
public class Test_4_2 {
```

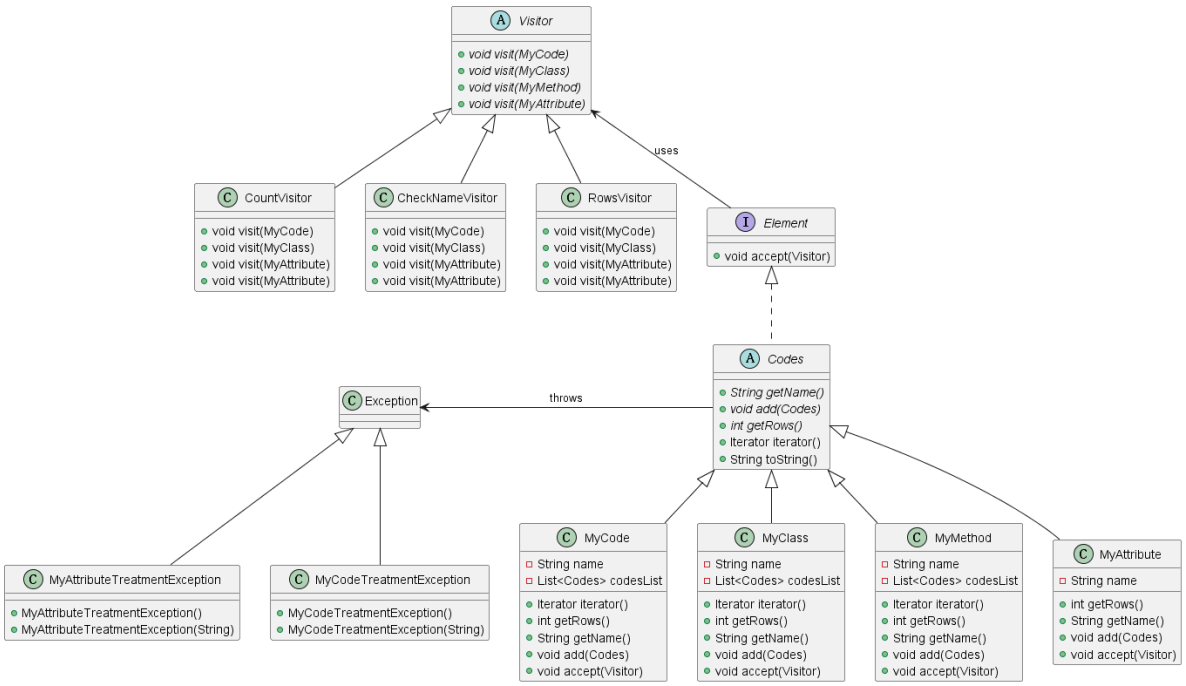
```
    public static void main(String[] args) {
        Display d1 = new Table("This Is A Table");
        d1.showTable();

        Display d2 = new Head(d1,'-');
        d2.showTable();

        Display d3 = new Tail(d2,'-');
        d3.showTable();

        Display d4 = new Tail(d3,'#');
        d4.showTable();

        Display d5 = new Head(d4,'*');
        d5.showTable();

    }
}
```

- Util.java

```
public class Util {
    protected static String MakeLine(int sz,Character ch){
        StringBuffer buf= new StringBuffer();
        for (int i=0;i<sz;i++){
            buf.append(ch);
        }
        return buf.toString();
    }
}
```

3. 访问者模式

类图



实现代码

- CheckNameVisitor.java

```
public class CheckNameVisitor extends Visitor{
    @Override
    public void visit(MyCode myCode) {
        System.out.println(myCode.toString() + ": Name is good");
    }

    @Override
    public void visit(MyClass myClass) {
        System.out.println(myClass.toString() + ": Name is good");
    }

    @Override
    public void visit(MyAttribute myAttribute) {
        System.out.println(myAttribute.toString() + ": Name is good");
    }

    @Override
    public void visit(MyMethod myMethod) {
        System.out.println(myMethod.toString() + ": Name is good");
    }
}
```

- Codes.java

```
import java.util.Iterator;

public abstract class Codes implements Element{
    public abstract String getName();
    public abstract void add(Codes codes) throws MyAttributeTreatmentException;
    public Iterator iterator() throws MyAttributeTreatmentException{
        throw new MyAttributeTreatmentException();
    }
    public String toString(){
        return " ["+this.getClass().getName()+"] " + getName();
    }
    public abstract int getRows() throws
MyCodeTreatmentException,MyAttributeTreatmentException;
}
```

- CountVisitor.java

```
import java.util.Iterator;

public class CountVisitor extends Visitor{
    private String curPrefix = "";
    @Override
    public void visit(MyCode myCode) {
        Iterator it = myCode.iterator();
        int cnt = 0;
        while(it.hasNext()){
```

```
            cnt++;
            it.next();
        }
        System.out.println(myCode.toString() + ": has " + cnt + " Class(es)");
    }

    @Override
    public void visit(MyClass myClass) {
        Iterator it = myClass.iterator();
        int cntMethod = 0 , cntAttribute = 0;
        while(it.hasNext()){
            Codes codes =(Codes) it.next();
            if ( ((String)codes.getClass().getName()).equals("MyMethod"))
cntMethod++;
            else cntAttribute++;
        }
        System.out.println(myClass.toString() + "has " + cntMethod + "
Method(s)");
        System.out.println(myClass.toString() + "has " + cntAttribute + "
Attribute(s)");
    }

    @Override
    public void visit(MyAttribute myAttribute) {
        System.out.println(myAttribute.toString() + "has nothing to be count ");
    }

    @Override
    public void visit(MyMethod myMethod) {
        System.out.println(myMethod.toString() + "has nothing to be count ");
    }

}
```

- Element.java

```
public interface Element {
    public void accept(Visitor visitor);
}
```

- MyAttribute.java

```
public class MyAttribute extends Codes {
    private String name;

    public MyAttribute(String name){
        this.name=name;
    }

    @Override
    public int getRows() throws MyCodeTreatmentException,
MyAttributeTreatmentException {
        throw new MyAttributeTreatmentException();
    }
```

```
    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    public void add(Codes codes) throws MyAttributeTreatmentException{
        throw new MyAttributeTreatmentException();
    }
}
```

- MyAttributeTreatmentException.java

```
public class MyAttributeTreatmentException extends Exception{
    public MyAttributeTreatmentException(){};
    public MyAttributeTreatmentException(String msg){
        super(msg);
    }

}
```

- MyClass.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MyClass extends Codes {
    private String name;
    private int rows;
    private List<Codes> codesList = new ArrayList<Codes>();

    public MyClass(String name,int rows){
        this.name=name;
        this.rows=rows;
    }

    public Iterator iterator(){
        return codesList.iterator();
    }

    @Override
    public int getRows() throws MyCodeTreatmentException,
MyAttributeTreatmentException {
        return rows;
    }

    @Override
    public String getName() {
        return this.name;
    }
```

```
    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    public void add(Codes codes) {
        codesList.add(codes);
    }
}
```

- MyCode.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MyCode extends Codes{
    private String name;
    private List<Codes> codesList = new ArrayList<Codes>();

    public MyCode(String name){
        this.name=name;
    }

    public Iterator iterator(){
        return codesList.iterator();
    }

    @Override
    public int getRows() throws MyCodeTreatmentException,
MyAttributeTreatmentException {
        throw new MyCodeTreatmentException();
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public void add(Codes codes) {
        codesList.add(codes);
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}
```

- MyCodeTreatmentException.java

```
public class MyCodeTreatmentException extends Exception{
    public MyCodeTreatmentException(){}
    public MyCodeTreatmentException(String msg){
        super(msg);
    }
}
```

- MyMethod.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MyMethod extends Codes {
    private String name;
    private int rows;
    private List<Codes> codesList = new ArrayList<Codes>();

    public MyMethod(String name,int rows){
        this.name=name;
        this.rows=rows;
    }

    public Iterator iterator(){
        return codesList.iterator();
    }

    @Override
    public int getRows() throws MyCodeTreatmentException,
MyAttributeTreatmentException {
        return rows;
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    public void add(Codes codes) {
        codesList.add(codes);
    }
}
```

- RowsVisitor.java

```
import java.util.Iterator;

public class RowsVisitor extends Visitor{
    @Override
    public void visit(MyCode myCode) {
        int cnt = 0;
```

```
        Iterator it = myCode.iterator();
        while(it.hasNext()){
            Codes codes =(Codes) it.next();
            try{
                cnt += codes.getRows();
            } catch (Exception e){
                e.printStackTrace();
            }
        }
        System.out.println(myCode.toString() + ": has "+ cnt + " line(s)");
    }

    @Override
    public void visit(MyClass myClass) {
        int cnt = 0;
        try {
            cnt = myClass.getRows();
        } catch (Exception e){
            e.printStackTrace();
        }
        System.out.println(myClass.toString() + ": has "+cnt + " line(s)");
    }

    @Override
    public void visit(MyAttribute myAttribute) {
        System.out.println(myAttribute.toString() + ": has only 1 line");
    }

    @Override
    public void visit(MyMethod myMethod) {
        int cnt = 0;
        try {
            cnt = myMethod.getRows();
        } catch (Exception e){
            e.printStackTrace();
        }
        System.out.println(myMethod.toString() + ": has "+cnt + " line(s)");
    }
}
```

- Test_4_3.java

```
import java.util.concurrent.Callable;

public class Test_4_3 {
    public static void main(String[] args) {
        Codes codeRoot = new MyCode("root");
        Codes class1 = new MyClass("class1",10);
        Codes class2 = new MyClass("class2",8);
        Codes method1 = new MyMethod("method1",4);
        Codes method2 = new MyMethod("method2",5);
        Codes method3 = new MyMethod("method3",8);
        Codes attribute1 = new MyAttribute("attribute1");
        Codes attribute2 = new MyAttribute("attribute2");
        Codes attribute3 = new MyAttribute("attribute3");
        Codes attribute4 = new MyAttribute("attribute4");
        try{
```

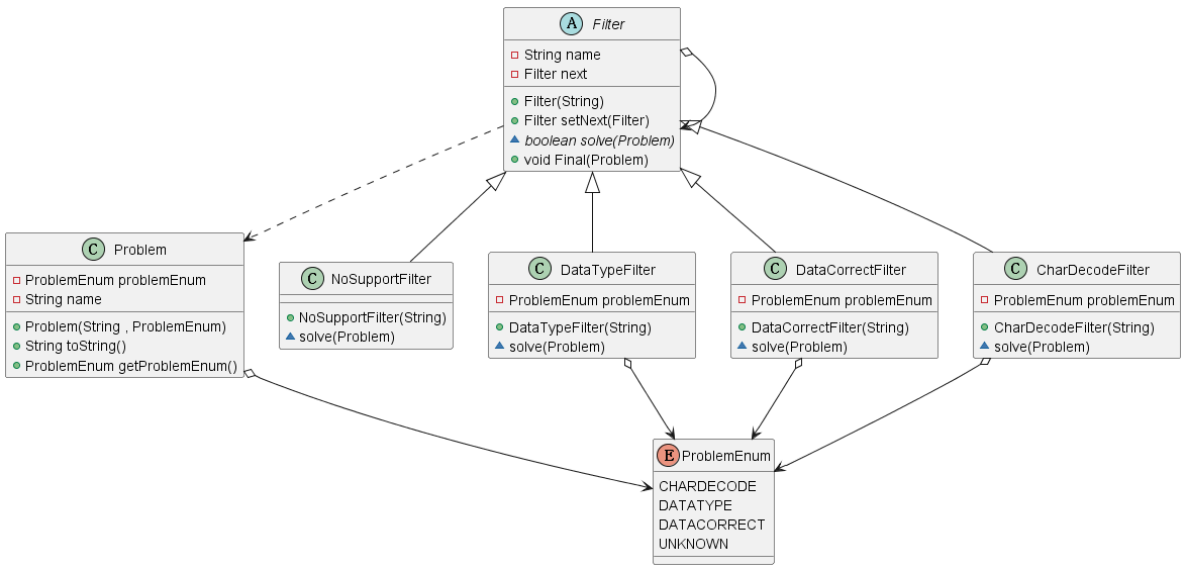
```
            codeRoot.add(class1);
            codeRoot.add(class2);
            class1.add(method1);
            class1.add(method3);
            class2.add(method2);
            class1.add(attribute4);
            class2.add(attribute1);
            class2.add(attribute2);
            class2.add(attribute3);
        } catch (Exception e){
            e.printStackTrace();
        }
        codeRoot.accept(new RowsVisitor());
        codeRoot.accept(new CountVisitor());
        class1.accept(new CheckNameVisitor());
        class1.accept(new CountVisitor());
        class2.accept(new CountVisitor());
        attribute1.accept(new CheckNameVisitor());
        method2.accept(new CheckNameVisitor());
        method1.accept(new RowsVisitor());
        attribute4.accept(new RowsVisitor());
    }
}
```

- Visitor.java

```
public abstract class Visitor {
    public abstract void visit(MyCode myCode);
    public abstract void visit(MyClass myClass);
    public abstract void visit(MyAttribute myAttribute);
    public abstract void visit(MyMethod myMethod);
}
```

4. 职责链模式

类图



实现代码

- CharDecodeFilter.java

```
public class CharDecodeFilter extends Filter{

    private ProblemEnum problemEnum = ProblemEnum.CHARDECODE;

    public CharDecodeFilter(String name) {
        super(name);
    }

    @Override
    protected boolean solve(Problem problem) {
        return problem.getProblemEnum().equals(this.problemEnum);
    }
}
```

- DataCorrectFilter.java

```
public class DataCorrectFilter extends Filter{

    private ProblemEnum problemEnum = ProblemEnum.DATACORRECT;

    public DataCorrectFilter(String name) {
        super(name);
    }

    @Override
    protected boolean solve(Problem problem) {
        return problem.getProblemEnum().equals(this.problemEnum);
    }
}
```

- DataTypeFilter.java

```
public class DataTypeFilter extends Filter{

    private ProblemEnum problemEnum = ProblemEnum.DATATYPE;

    public DataTypeFilter(String name) {
        super(name);
    }

    @Override
    protected boolean solve(Problem problem) {
        return problem.getProblemEnum().equals(this.problemEnum);
    }
}
```

- Filter.java

```
public abstract class Filter {
    private String name;
    private Filter next=null;
```

```
    public Filter(String name){
        this.name= name;
    }
    public Filter setNext(Filter filter){
        this.next=filter;
        return filter;
    }

    protected abstract boolean solve(Problem problem);

    public void Final(Problem problem){
        if(solve(problem)){
            System.out.println(problem.toString() + " Has Been Solved by "+
this.getClass().getName());
        } else if(next!=null){
            next.Final(problem);
        } else {
            System.out.println(problem.toString() + " Cannot be solved by Any
Filter");
        }
    }

}
```

- NoSupportFilter.java

```
public class NoSupportFilter extends Filter{

    public NoSupportFilter(String name) {
        super(name);
    }

    @Override
    protected boolean solve(Problem problem) {
        return false;
    }
}
```

- Problem.java

```
public class Problem {
    private ProblemEnum problemEnum;
    private String name;
    public Problem(String name,ProblemEnum problemEnum){
        this.name=name;
        this.problemEnum=problemEnum;
    }

    public String toString(){
        return "[ " + this.problemEnum.toString() + "Problem: "+this.name+ " ]";
    }

    public ProblemEnum getProblemEnum(){
        return this.problemEnum;
    }
}
```

- ProblemEnum.java

```
public enum ProblemEnum {
    CHARDECODE,DATATYPE,DATACORRECT,UNKNOWN
}
```

- Test_4_4.java

```
public class Test_4_4 {
    public static void main(String[] args) {
        Filter Emet = new CharDecodeFilter("Emet");
        Filter Memet = new DataCorrectFilter("Memet");
        Filter Semet = new DataTypeFilter("Semet");
        Filter Appah = new NoSupportFilter("Appah");
        Problem Oghri = new Problem("Oghri",ProblemEnum.DATACORRECT);
        Problem Qaraqchi = new Problem("Qaraqchi",ProblemEnum.CHARDECODE);
        Problem Munapiq = new Problem("Munapiq",ProblemEnum.DATATYPE);
        Problem Satqin = new Problem("Sarqin",ProblemEnum.UNKNOWN);
        Appah.setNext(Emet).setNext(Memet).setNext(Semet);
        Appah.Final(Oghri);
        Appah.Final(Qaraqchi);
        Appah.Final(Munapiq);
        Appah.Final(Satqin);
    }
}
```

五、实验小结

本次实验学会了组合模式，装饰模式,访问者模式和职责链模式，在做这些作业的过程中，不仅对这5类设计模式有了直接的理解和体会，同时对java代码更加熟悉了，也更深入的了解了面向对象的编程的思想，对代码整体的设计的把握也更得心应手了