天津大学

设计模式（4）实验报告

学　　院　智算学部

专　　业　软件工程

学　　号　3019213043

姓　　名　刘京宗

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

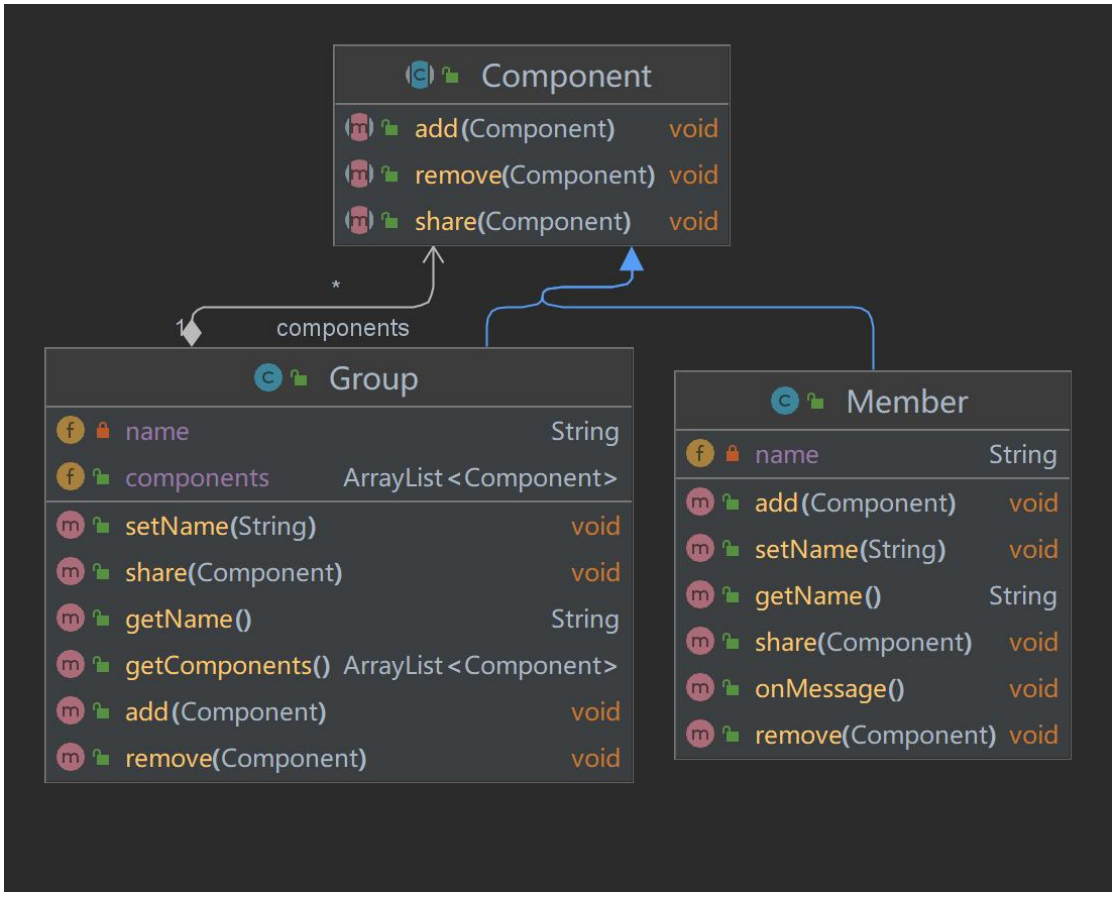
在某 Web 框架中采用职责链模式来组织数据过滤器，不同的数据过滤器提供了不同的功能，例如字符编码转换过滤器、数据类型转换过滤器、数据校验过滤器等，可以将多个过

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。

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。

4.1 组合模式

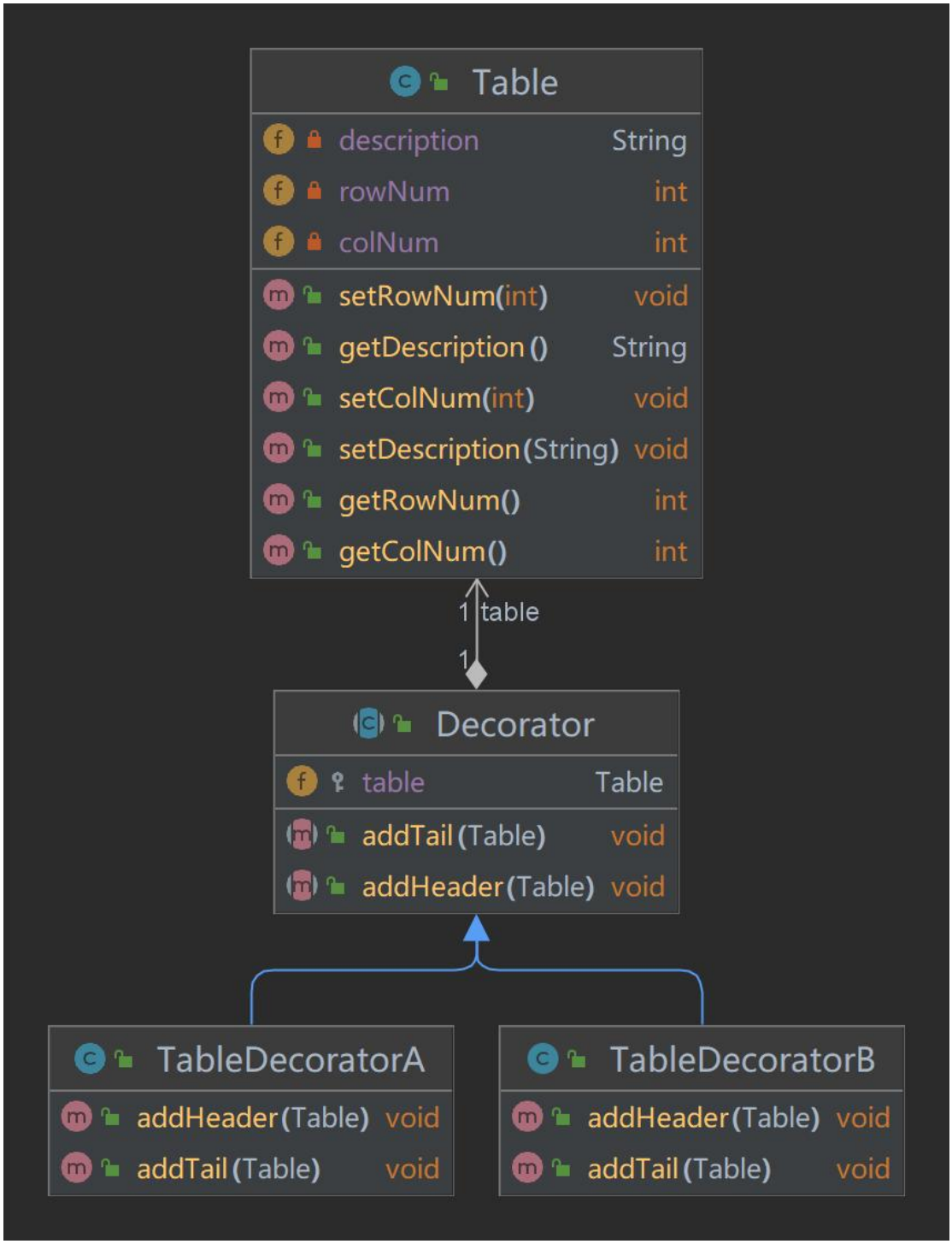



```
public abstract class Component {
    public abstract void add(Component component);

    public abstract void remove(Component component);

    public abstract void share(Component component);
}
public class Group extends Component {
    public Group(String name) {
        this.name = name;
    }

    public ArrayList<Component> getComponents() {
        return components;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private String name;
    public ArrayList<Component> components = new ArrayList<>();

    @Override
    public void add(Component ccomponent) {
        components.add(ccomponent);
    }

    @Override
    public void remove(Component component) {
        components.remove(component);
    }

    @Override
    public void share(Component component) {
        if (component instanceof Group) {
            ArrayList<Component> com = ((Group) component).getComponents();
            for (Component object : components) {
                object.share(object);
            }
        } else if (component instanceof Member) {
            ((Member) component).onMessage();
        }
    }
}
public class Member extends Component {
    public Member(String name) {
        this.name = name;

    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    private String name;
    @Override
    public void add(Component ccomponent) {
        System.out.println("调这个函数也没用");
    }

    @Override
    public void remove(Component component) {
        System.out.println("调这个函数也没用");
    }

    @Override
    public void share(Component component) {
        if (component instanceof Group) {
            ArrayList<Component> components = ((Group) component).getComponents();
            if (components.contains(this)) {
                components.remove(this);
            }
            for (Component obj : components) {
                obj.share(obj);
            }
            components.add(this);
        } else if (component instanceof Member) {
            ((Member) component).onMessage();
        }
    }

    public void onMessage() {
        System.out.println(this.getName() + "收到消息");
    }

}
```

4.2 装饰模式

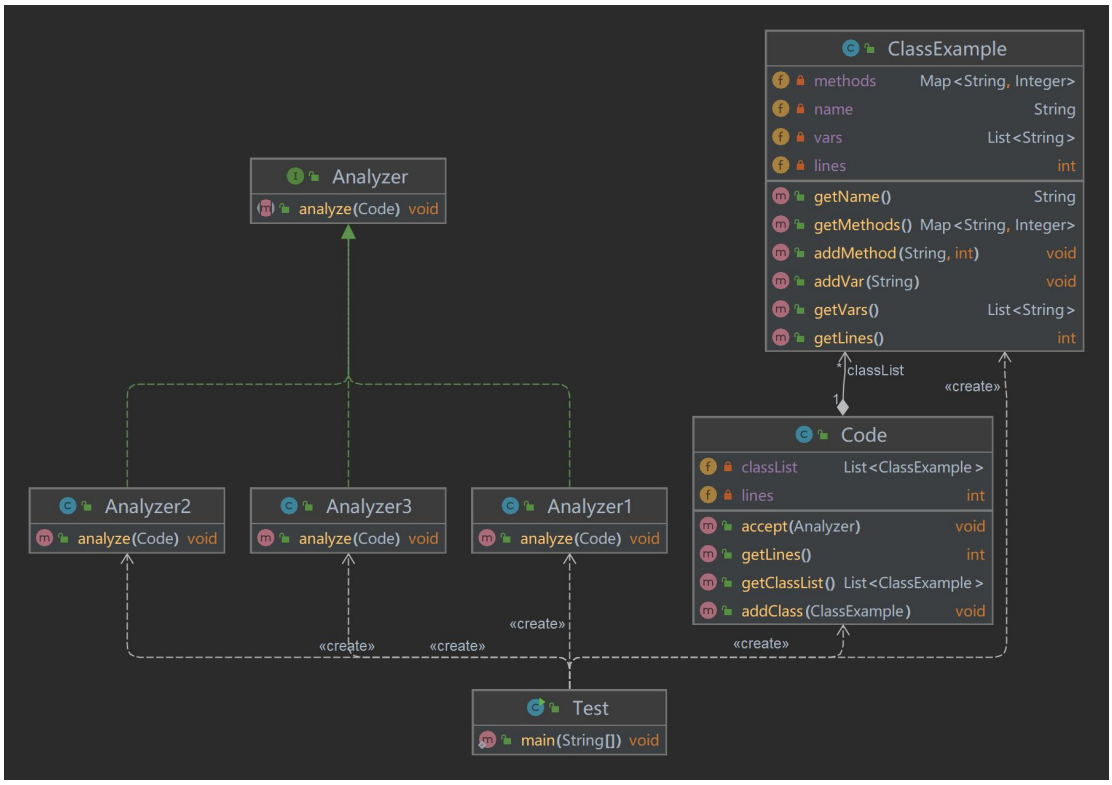



```
public abstract class Decorator {
    protected Table table;

    public abstract void addHeader(Table table);

    public abstract void addTail(Table table);
}
public class Table {
    private int rowNum;
```

```
    private int colNum;
    private String description;

    public int getRowNum() {
        return rowNum;
    }

    public void setRowNum(int rowNum) {
        this.rowNum = rowNum;
    }

    public int getColNum() {
        return colNum;
    }

    public void setColNum(int colNum) {
        this.colNum = colNum;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
public class TableDecoratorA extends Decorator {
    @Override
    public void addHeader(Table table) {
        System.out.println("TableDecoratorA addHeader");
    }

    @Override
    public void addTail(Table table) {
        System.out.println("TableDecoratorA addTail");
    }
}
public class TableDecoratorB extends Decorator {
    @Override
    public void addHeader(Table table) {
        System.out.println("TableDecoratorB addHeader");
    }
```

```
    @Override
    public void addTail(Table table) {
        System.out.println("TableDecoratorB addHeader");
    }
}
```

4.3 访问者模式



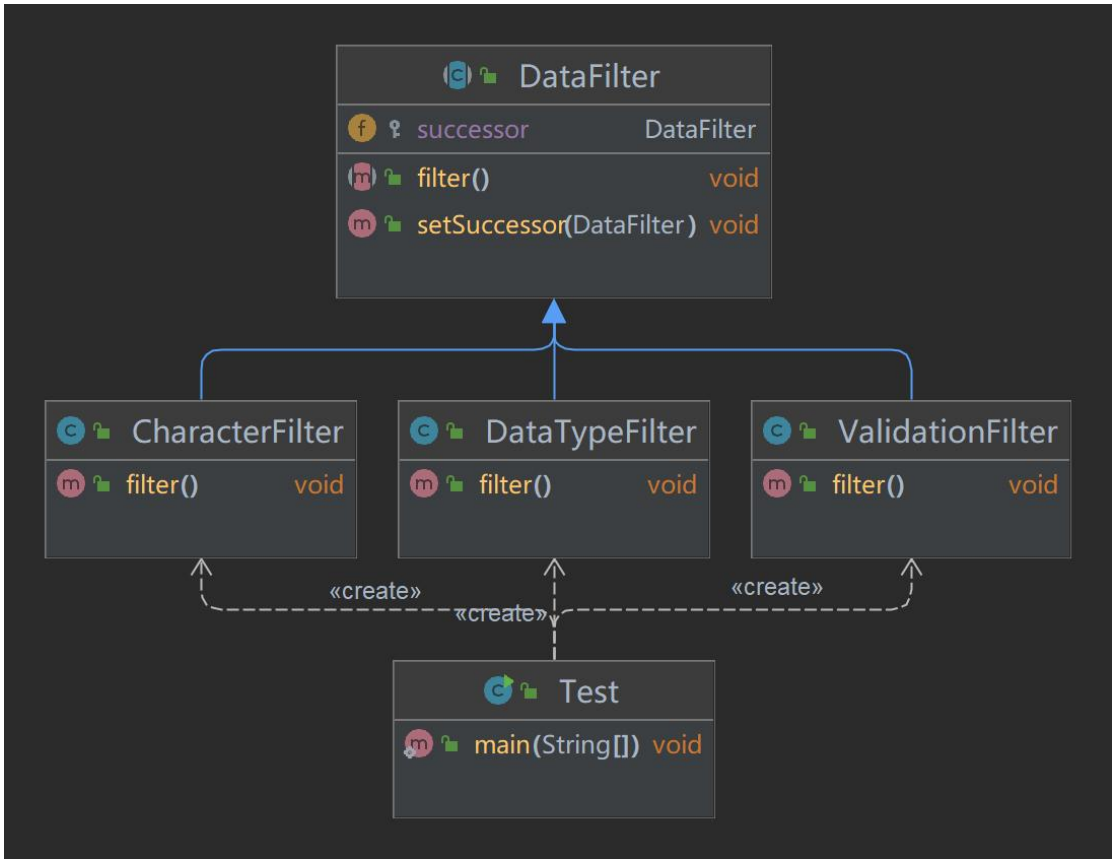

```
public interface Analyzer {
    void analyze(Code code);
}
public class Analyzer1 implements Analyzer {

    @Override
    public void analyze(Code code) {
        System.out.println("经分析代码中类的总个数：" + code.getClassList().size());
        for (ClassExample classExample : code.getClassList()) {
            System.out.println(classExample.getName() + "类拥有的属性个数：" +
classExample.getVars().size());
            System.out.println(classExample.getName() + "类拥有的方法个数：" +
classExample.getMethods().size());
        }
    }
}
public class Analyzer2 implements Analyzer {

    @Override
```

```
    public void analyze(Code code) {
        System.out.print("经分析代码中声明的类有：");
        for (ClassExample classExample : code.getClassList()) {
            System.out.print(classExample.getName() + ", ");
        }
        System.out.print("\n");
        for (ClassExample classExample : code.getClassList()) {
            System.out.println(classExample.getName() + "类中定义的方法有：" +
classExample.getMethods().keySet());
            System.out.println(classExample.getName() + "类中定义的属性有" +
classExample.getVars());
        }
    }
}
public class Analyzer3 implements Analyzer {

    @Override
    public void analyze(Code code) {
        System.out.println("经分析代码总行数为：" + code.getLines());
        for (ClassExample classExample : code.getClassList()) {
            System.out.println(classExample.getName() + "类代码总行数：" +
classExample.getLines());
            for (String s : classExample.getMethods().keySet()) {
                System.out.println(classExample.getName() + "类中方法" + s + "()代码总行
数：" + classExample.getMethods().get(s));
            }
        }
    }
}
public class ClassExample {
    private String name;
    //自动计算行数
    private int lines;
    private List<String> vars = new ArrayList<String>();
    private Map<String, Integer> methods = new HashMap<String, Integer>();

    public ClassExample(String name) {
        this.name = name;
        this.lines = 0;
    }

    //变量声明单独占一行
    public void addVar(String var) {
        vars.add(var);
```

```
            lines += 1;
        }

        //方法行数直接给出
        public void addMethod(String method, int lines) {
            methods.put(method, lines);
            this.lines += lines;
        }

        public String getName() {
            return name;
        }

        public int getLines() {
            return lines;
        }

        public List<String> getVars() {
            return vars;
        }

        public Map<String, Integer> getMethods() {
            return methods;
        }
}
public class Code {
        private int lines;
        private List<ClassExample> classList = new ArrayList<ClassExample>();

        public int getLines() {
            return lines;
        }

        public List<ClassExample> getClassList() {
            return classList;
        }

        public void addClass(ClassExample classExample) {
            classList.add(classExample);
            lines += classExample.getLines();
        }

        public void accept(Analyzer analyzer) {
            analyzer.analyze(this);
```

```
    }
}
public class Test {
    public static void main(String[] args) {
        ClassExample class1 = new ClassExample("A");
        class1.addVar("name");
        class1.addVar("ID");
        class1.addMethod("changeName", 24);
        class1.addMethod("changeID", 68);

        ClassExample class2 = new ClassExample("B");
        class2.addVar("sex");
        class2.addVar("age");
        class2.addVar("height");
        class2.addVar("weight");
        class2.addMethod("changeSex", 109);
        class2.addMethod("intro", 20);
        class2.addMethod("LoseWeight", 47);

        ClassExample class3 = new ClassExample("C");
        class3.addVar("size");
        class3.addMethod("getSize", 10);

        Code code = new Code();
        code.addClass(class1);
        code.addClass(class2);
        code.addClass(class3);

        code.accept(new Analyzer1());
        code.accept(new Analyzer2());
        code.accept(new Analyzer3());
    }
}
```

4.4 职责链模式




```
public class CharacterFilter extends DataFilter {
    public void filter() {
        System.out.println("字符编码转换过滤器");
        this.successor.filter();
    }
}
public abstract class DataFilter {
    protected DataFilter successor;

    public void setSuccessor(DataFilter successor) {
        this.successor = successor;
    }

    public abstract void filter();

}
public class DataTypeFilter extends DataFilter {
    public void filter() {
        System.out.println("调用数据类型转换过滤器");
        this.successor.filter();
    }
}
```

```
public class Test {
	public static void main(String[] args) {
		DataFilter df1 = new CharacterFilter();
		DataFilter df2 = new DataTypeFilter();
		DataFilter df3 = new ValidationFilter();

		df1.setSuccessor(df2);
		df2.setSuccessor(df3);
		df1.filter();
	}
}
public class ValidationFilter extends DataFilter {
	public void filter() {
		System.out.println("调用数据校验过滤器");
	}
}
```

五、实验小结

通过这次实验，我熟悉了组合模式、装饰模式、访问者模式、职责链模式、这些设计模式，收益匪浅。