设计模式实验 13-16

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. 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。

2. 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。

3. 外观模式：某软件公司为新开发的智能手机控制与管理软件提供了一键备份

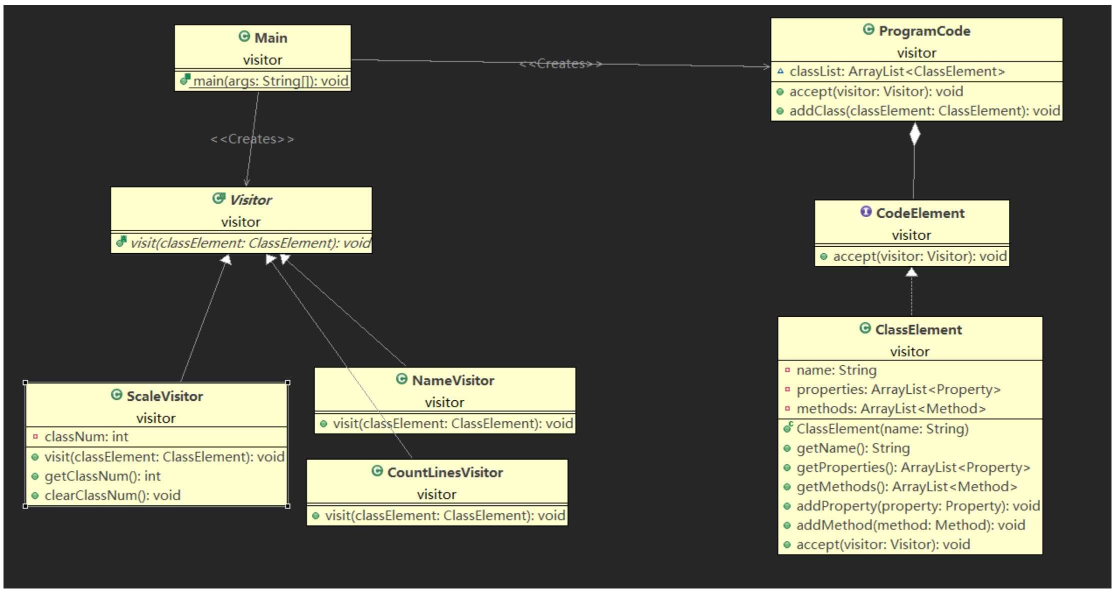
功能，通过该功能可以 将原本存储在手机中的通讯录、短信、照片、歌曲等资料一次性全部拷贝到移动存储个质（例 如 MMC 卡或 SD 卡）中。在实现过程中需要与多个已有的类进行交互，例如通讯录管理 类、短信管理类等。为了降低系统的耦合度，试使用外观模式来设计并编程模拟实现该一键 备份功能。

4. 中介者模式：为了大力发展旅游业，某城市构建了一个旅游综合信息系统，该系统包括旅行社子系统 （ravel companies Subsystem）、宾馆子系统（HotelsSubsystem）、餐厅于系统(Restaurants Subsystem)、机场子系统（Airport Subsystem）、旅游景点子系统（Tourism Attractions Subsystem）等多个子系统，通过该旅游综合信息系统，各类企业之间可实现信息共享，一 家企业可以将客户信息传递给其他合作伙伴。例如，当一家旅行社有一些客户后，该旅行社 可以将客户信息传送到宾馆子系统、餐厅子系统、机场子系统和旅游景点子系统；宾馆也可 以将顾客信息传送到旅行社子系统、餐子系统、机场子系统和旅游景点子系统；机场也可以 将乘客信息传送到旅行社子系统、宾馆子系统、餐厅子系统和旅游景点子系统。由于这些子 系统之间存在较为复杂的交互关系，现采用中介者模式为该旅游综合信息系统提供一个高层 设计，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. 访问者模式


Main
visitor
main(args: String[]): void
<<Creates>>
ProgramCode
visitor
classList: ArrayList<ClassElement>
accept(visitor: Visitor): void
addClass(classElement: ClassElement): void
<<Creates>>
Visitor
visitor
visit(classElement: ClassElement): void
CodeElement
visitor
accept(visitor: Visitor): void
ClassElement
visitor
name: String
properties: ArrayList<Property>
methods: ArrayList<Method>
ClassElement(name: String)
getName(): String
getProperties(): ArrayList<Property>
getMethods(): ArrayList<Method>
addProperty(property: Property): void
addMethod(method: Method): void
accept(visitor: Visitor): void
ScaleVisitor
visitor
classNum: int
visit(classElement: ClassElement): void
getClassNum(): int
clearClassNum(): void
NameVisitor
visitor
visit(classElement: ClassElement): void
CountLinesVisitor
visitor
visit(classElement: ClassElement): void

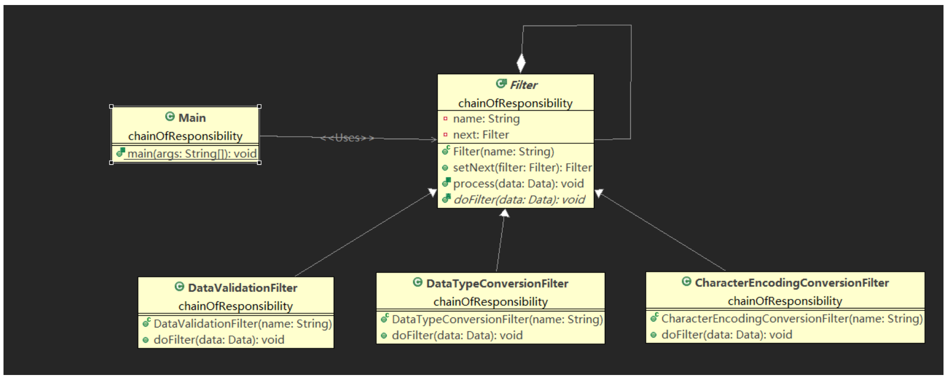
代码实现:

```java
public interface CodeElement {

    public void accept(Visitor visitor);
}

public class ClassElement implements CodeElement{

    private String name;
    private ArrayList<Property> properties = new ArrayList<>();
    private ArrayList<Method> methods = new ArrayList<>();

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

    public String getName() {
        return name;
    }

    public ArrayList<Property> getProperties(){
        return properties;
    }

    public ArrayList<Method> getMethods(){
        return methods;
    }

    public void addProperty(Property property) {
        properties.add(property);
    }

    public void addMethod(Method method) {
        methods.add(method);
    }


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

    }
}

public class Property{

    private String name;

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

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

```java
public class Method{

    private String name;
    private int lineNum;

    public Method(String name, int lineNum) {
        this.name = name;
        this.lineNum = lineNum;
    }

    public String getName() {
        return name;
    }

    public int getLineNum() {
        return lineNum;
    }

}

public abstract class Visitor {

    public abstract void visit(ClassElement classElement);

}

public class ScaleVisitor extends Visitor{

    private int classNum = 0;


    @Override
    public void visit(ClassElement classElement) {
        classNum++;
        int propertyNum = 0;
        int methodNum = 0;
        ArrayList<Property> properties = classElement.getProperties();
        ArrayList<Method> methods = classElement.getMethods();
        for(Property property : properties) {
            propertyNum++;
        }
        for(Method method: methods) {
            methodNum++;
        }
        System.out.println("Class " + classElement.getName() + "'s number of 
properties is " + propertyNum + " and number of methods is " + methodNum);

    }


    public int getClassNum() {
        return classNum;
    }

    public void clearClassNum() {
        classNum = 0;
    }
}
```

```java
public class NameVisitor extends Visitor{


    @Override
    public void visit(ClassElement classElement) {
        String className = classElement.getName();
        ArrayList<Property> properties = classElement.getProperties();
        ArrayList<Method> methods = classElement.getMethods();
        System.out.println("Class " + className + " has properties and methods as
follows");
        System.out.println("Properties: ");
        for(Property property : properties) {
            System.out.println(property.getName());
        }
        System.out.println("Methods: ");
        for(Method method: methods) {
            System.out.println(method.getName());
        }
    }
}

public class CountLinesVisitor extends Visitor{

    @Override
    public void visit(ClassElement classElement) {
        int classLine = 0;
        String className = classElement.getName();
        ArrayList<Property> properties = classElement.getProperties();
        ArrayList<Method> methods = classElement.getMethods();
        for(Property property : properties) {
            classLine++;
        }
        for(Method method: methods) {
            classLine += method.getLineNum();
        }

        System.out.println("Class " + className + " has " + classLine +" lines.");

    }
}

public class ProgramCode {

    ArrayList<ClassElement> classList = new ArrayList<>();

    public void accept(Visitor visitor) {
        for(ClassElement classElement: classList) {
            visitor.visit(classElement);
        }
    }

    public void addClass(ClassElement classElement) {
        classList.add(classElement);
    }

}

public class Main {
    public static void main(String[] args) {

        ClassElement a = new ClassElement("a");
        Property pa = new Property("pa");
        Method ma = new Method("ma", 10);
        a.addMethod(ma);
        a.addProperty(pa);

        ClassElement b = new ClassElement("b");
        Property pb = new Property("pb");
        Method mb = new Method("mb", 100);
        Property pc = new Property("pc");
        Method mc = new Method("mc", 1);
        Property pd = new Property("pd");
        b.addMethod(mb);
        b.addMethod(mc);

        b.addProperty(pb);
        b.addProperty(pc);
        b.addProperty(pd);

        ProgramCode pcode = new ProgramCode();
        pcode.addClass(a);
        pcode.addClass(b);

        ScaleVisitor sv = new ScaleVisitor();
        NameVisitor nv = new NameVisitor();
        CountLinesVisitor cv = new CountLinesVisitor();

        pcode.accept(sv);
        System.out.println("There are total " + sv.getClassNum() + " class.");
        sv.clearClassNum();
        pcode.accept(nv);
        pcode.accept(cv);



    }

}
```

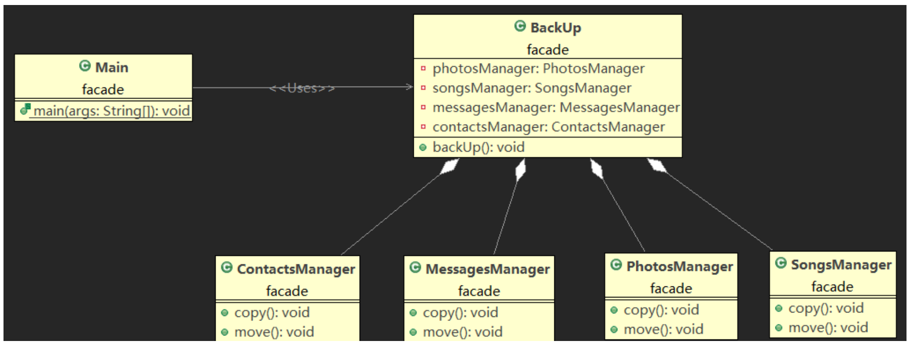
2. 职责链模式


Main
chainOfResponsibility
main(args: String[]): void
<<Uses>>
Filter
chainOfResponsibility
name: String
next: Filter
Filter(name: String)
setNext(filter: Filter): Filter
process(data: Data): void
doFilter(data: Data): void
DataValidationFilter
chainOfResponsibility
DataValidationFilter(name: String)
doFilter(data: Data): void
DataTypeConversionFilter
chainOfResponsibility
DataTypeConversionFilter(name: String)
doFilter(data: Data): void
CharacterEncodingConversionFilter
chainOfResponsibility
CharacterEncodingConversionFilter(name: String)
doFilter(data: Data): void

代码实现:
```java
public abstract class Filter {

    private String name;
    private Filter next;

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

    public Filter setNext(Filter filter) {
        next = filter;
        return next;
    }

    public final void process(Data data) {
        doFilter(data);
        if(next != null) {
            next.process(data);
        }
        else {
            System.out.println("The process is finished.");
        }

    }

    public abstract void doFilter(Data data);

}

public class DataValidationFilter extends Filter{

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

    @Override
    public void doFilter(Data data) {
        System.out.println("Do Data Validation Filter to " + data.getName());

    }

}

public class DataTypeConversionFilter extends Filter{

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

    @Override
    public void doFilter(Data data) {
        System.out.println("Do Data Type Conversion to " + data.getName());
    }

}
```

```
public class CharacterEncodingConversionFilter extends Filter{

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

    @Override
    public void doFilter(Data data) {
        System.out.println("Do Character Encoding Conversion to " +
data.getName());

    }

}

public class Data {

    private String name;

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

    public String getName() {
        return name;
    }

}

public class Main {

    public static void main(String[] args) {
        Filter a = new CharacterEncodingConversionFilter("A");
        Filter b = new DataTypeConversionFilter("B");
        Filter c = new DataValidationFilter("C");
        Filter d = new DataValidationFilter("C");

        Data data = new Data("map");
        a.setNext(b).setNext(c).setNext(d);
        a.process(data);
    }

}
```

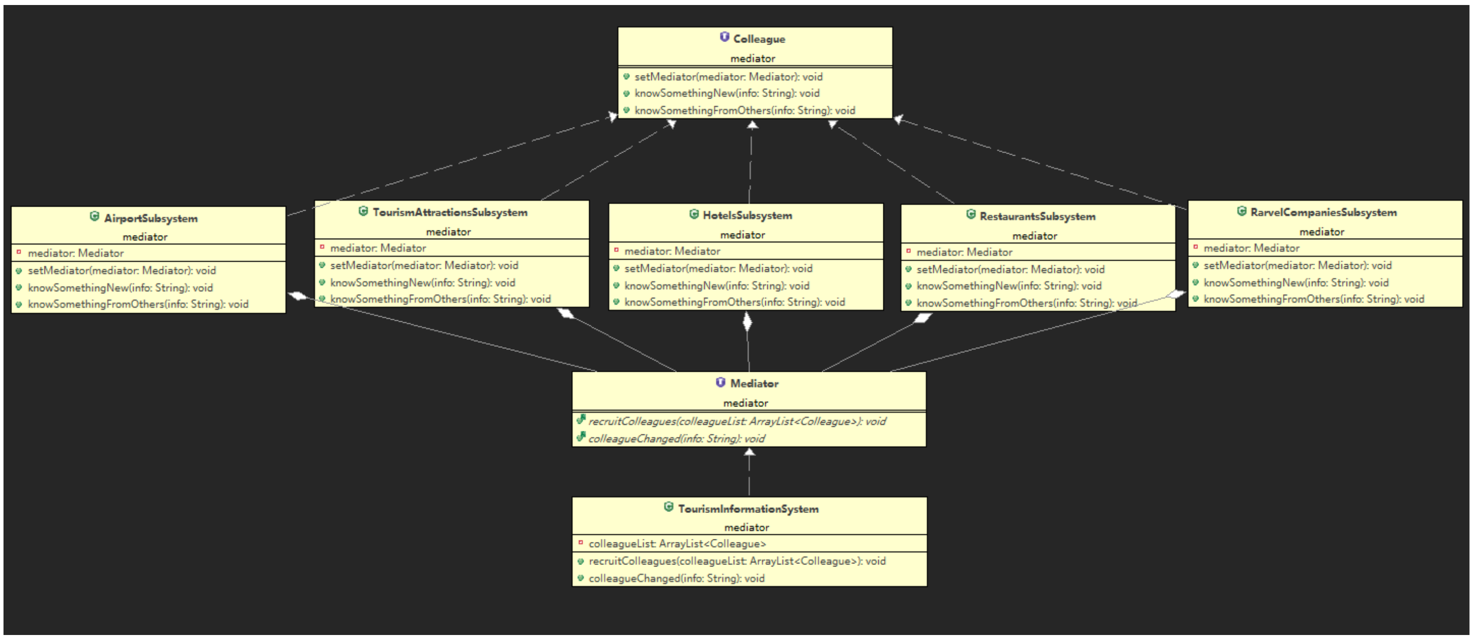
3. 外观模式



代码实现:
```java
public class BackUp {

    private PhotosManager photosManager = new PhotosManager();
    private SongsManager songsManager = new SongsManager();
    private MessagesManager messagesManager = new MessagesManager();
    private ContactsManager contactsManager = new ContactsManager();

    public void backUp() {
        photosManager.copy();
        photosManager.move();
        songsManager.copy();
        songsManager.move();
        messagesManager.copy();
        messagesManager.move();
        contactsManager.copy();
        contactsManager.move();
    }


}

public class ContactsManager {

    public void copy() {
        System.out.println("Contacts have been copied.");
    }

    public void move() {
        System.out.println("Contacts have been moved.");
    }

}

public class MessagesManager {

    public void copy() {
        System.out.println("Messages have been copied.");
    }

    public void move() {
        System.out.println("Messages have been moved.");
    }

}

public class PhotosManager {

    public void copy() {
        System.out.println("Photos have been copied.");
    }

    public void move() {
        System.out.println("Photos have been moved.");
    }

}
```

```java
public class SongsManager {

    public void copy() {
        System.out.println("Songs have been copied.");
    }

    public void move() {
        System.out.println("Songs have been moved.");
    }

}

public class Main {

    public static void main(String[] args) {
        BackUp backUp = new BackUp();
        backUp.backUp();
    }

}
```

4. 中介者模式



Colleague
mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
AirportSubsystem
mediator
mediator: Mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
TourismAttractionsSubsystem
mediator
mediator: Mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
HotelsSubsystem
mediator
mediator: Mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
RestaurantsSubsystem
mediator
mediator: Mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
RarvelCompaniesSubsystem
mediator
mediator: Mediator
setMediator(mediator: Mediator): void
knowSomethingNew(info: String): void
knowSomethingFromOthers(info: String): void
Mediator
mediator
recruitColleagues(colleagueList: ArrayList<Colleague>): void
colleagueChanged(info: String): void
TourismInformationSystem
mediator
colleagueList: ArrayList<Colleague>
recruitColleagues(colleagueList: ArrayList<Colleague>): void
colleagueChanged(info: String): void

代码实现:

```java
public interface Mediator {

    public abstract void recruitColleagues(ArrayList<Colleague> colleagueList);
    public abstract void colleagueChanged(String info);

}

public class TourismInformationSystem implements Mediator{
    private ArrayList<Colleague> colleagueList;

    @Override
    public void recruitColleagues(ArrayList<Colleague> colleagueList) {
        this.colleagueList = colleagueList;
        for(Colleague colleague: colleagueList) {
            colleague.setMediator(this);
        }

    }

    @Override
    public void colleagueChanged(String info) {

        for(Colleague colleague: colleagueList) {
            colleague.knowSomethingFromOthers(info);
        }
    }

}

public interface Colleague {

    public void setMediator(Mediator mediator);
    public void knowSomethingNew(String info);
    public void knowSomethingFromOthers(String info);

}

public class AirportSubsystem implements Colleague{
    private Mediator mediator;

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void knowSomethingNew(String info) {
        mediator.colleagueChanged(info);
    }

    public void knowSomethingFromOthers(String info) {
        System.out.println("AirportSubsystem know " + info + " from others.");
    }

}

public class HotelsSubsystem implements Colleague{

    private Mediator mediator;

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void knowSomethingNew(String info) {
        mediator.colleagueChanged(info);
    }
    public void knowSomethingFromOthers(String info) {
        System.out.println("HotelsSubsystem know " + info + " from others.");
    }

}

public class RarvelCompaniesSubsystem implements Colleague{

    private Mediator mediator;

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void knowSomethingNew(String info) {
        mediator.colleagueChanged(info);
    }

    public void knowSomethingFromOthers(String info) {
        System.out.println("RarvelCompaniesSubsystem know " + info + " from 
others.");
    }

}

public class RestaurantsSubsystem implements Colleague{

    private Mediator mediator;

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void knowSomethingNew(String info) {
        mediator.colleagueChanged(info);
    }

    public void knowSomethingFromOthers(String info) {
        System.out.println("RestaurantsSubsystem know " + info + " from others.");
    }

}

public class TourismAttractionsSubsystem implements Colleague{

    private Mediator mediator;

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void knowSomethingNew(String info) {
        mediator.colleagueChanged(info);
    }

    public void knowSomethingFromOthers(String info) {
        System.out.println("TourismAttractionsSubsystem know " + info + " from 
others.");
```

```
    }

}

public class Main {

    public static void main(String[] args) {
        Mediator mediator = new TourismInformationSystem();
        RarvelCompaniesSubsystem rc = new RarvelCompaniesSubsystem();
        HotelsSubsystem h = new HotelsSubsystem();
        RestaurantsSubsystem r = new RestaurantsSubsystem();
        AirportSubsystem a = new AirportSubsystem();
        TourismAttractionsSubsystem t = new TourismAttractionsSubsystem();

        ArrayList<Colleague> colleagues = new ArrayList<>();
        colleagues.add(rc);
        colleagues.add(r);
        colleagues.add(h);
        colleagues.add(a);
        colleagues.add(t);

        mediator.recruitColleagues(colleagues);

        r.knowSomethingNew("NEWS");
    }

}
```

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，我加强了对访问者模式、职责链模式、外观模式，中介者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。

中介者模式中，遇到的问题：中介者内部产生的组员无法被外部知道调用，需要在外部传入。但是如果固定传入的组员参数，不满足开闭原则，不利于后续的扩展。
解决办法 ：可以传入 Colleague 类型的 List，在中介内部遍历列表，实现功能。