设计模式实验 9-12

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. 桥接模式：某手机美图 APP 软件支持多种不同的图像格式，例如 JPG、GIF、BMP 等常用图像格 式，同时提供了多种不同的滤镜对图像进行处理，例如木刻滤镜（Cutout）、模糊滤镜(Blur)、 锐化滤镜（Sharpen）、纹理滤镜（Texture）等。现采用桥接模式设计该 APP 软件，使得 该软件能够为多种图像格式提供一系列图像处理滤镜，同时还能够很方便地增加新的图像格 式和滤镜，绘制对应的类图并编程模拟实现。

2. 策略模式：在某云计算模拟平台中提供了多种虚拟机迁移算法，例如动态迁移算法中的 Pre - Copy （预拷贝）算法、 Post - Copy （后拷贝）算法、 CR / RT - Motion 算法等，用户可以灵 活地选择所需的虚拟机迁移算法，也可以方便地增加新算法。现采用策略模式进行设计，绘 制对应的类图并编程模拟实现。

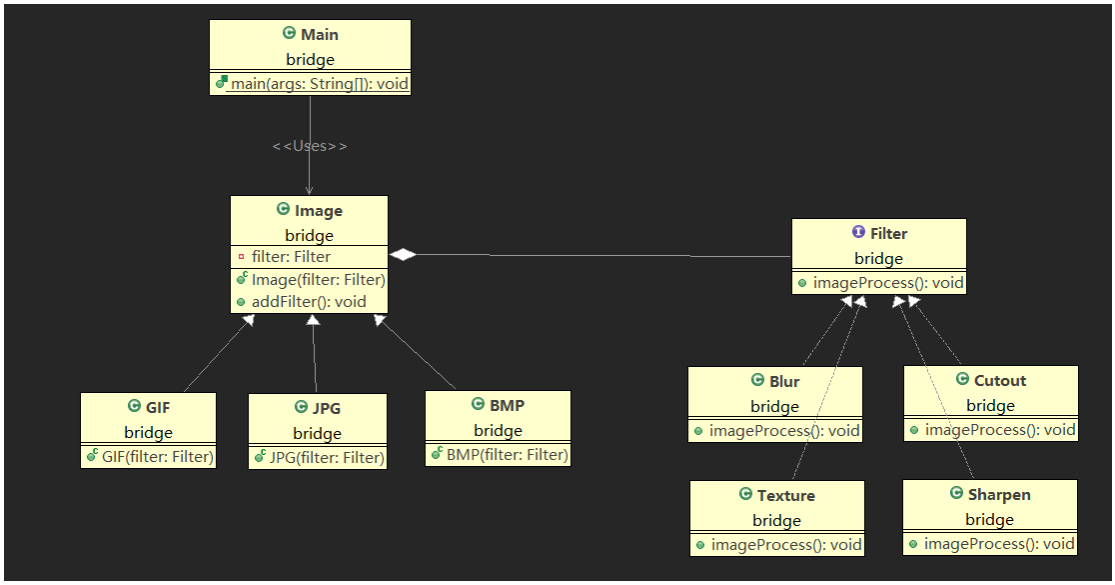
3. 组合模式：某移动社交软件要增加一个群组（Group）功能。通过设置，用户可以将自己的动态信 息（包括最新动态、新上传的视频以及分享的链接等）分享给某个特定的成员（Member）. 也可以分享给某个群组中的所有成员；用户可以将成员加至某个指定的群组；此外，还允许 用户在一个群组中加子群组，以便更加灵活地实现面向特定人群的信息共享。现采用组合模 式设计该群组功能，绘制对应的类图并编程模拟实现。

4. 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。

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.桥接模式



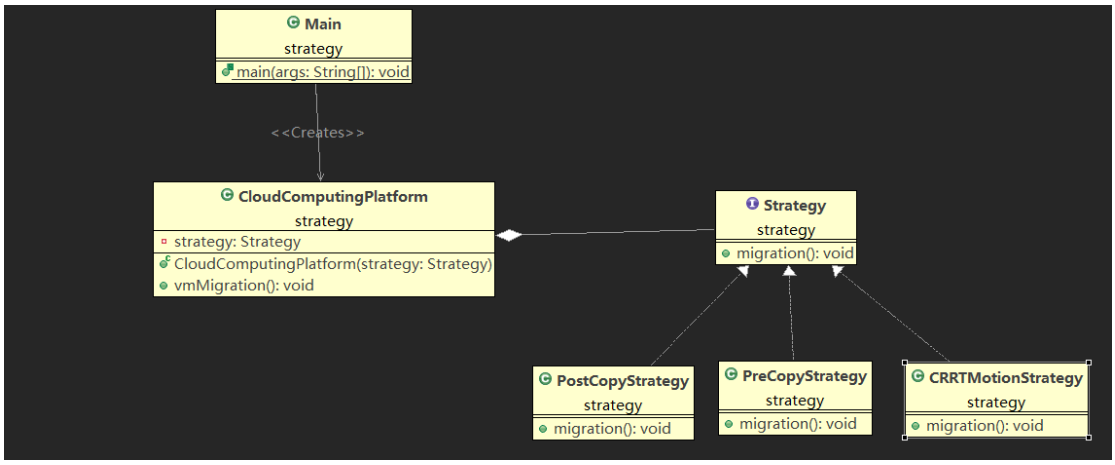
代码实现：

```
public interface Filter {

    public void imageProcess();

}

public class Cutout implements Filter{

    @Override
    public void imageProcess() {
        System.out.println("Adding Cutout Filter to image");

    }

}

public class Blur implements Filter{
```

```
    @Override
    public void imageProcess() {
        System.out.println("Adding Blur Filter to image");

    }

}

public class Sharpen implements Filter{

    @Override
    public void imageProcess() {
        System.out.println("Adding Sharpen Filter to image");

    }

}

public class Texture implements Filter{

    @Override
    public void imageProcess() {
        System.out.println("Adding Texture Filter to image");

    }

}

public class Image {

    private Filter filter;

    public Image(Filter filter) {
        this.filter = filter;
    }

    public void addFilter() {
        filter.imageProcess();
    }

}

public class JPG extends Image{

    public JPG(Filter filter) {
        super(filter);
    }
}

public class BMP extends Image{

    public BMP(Filter filter) {
        super(filter);
    }

}

public class GIF extends Image{
```

```
    public GIF(Filter filter) {
        super(filter);
    }

}

public class Main {

    public static void main(String[] args) {

        Image image1 = new JPG(new Cutout());
        Image image2 = new GIF(new Sharpen());
        Image image3 = new BMP(new Texture());

        image1.addFilter();
        image2.addFilter();
        image3.addFilter();

    }
}
```

2. 策略模式

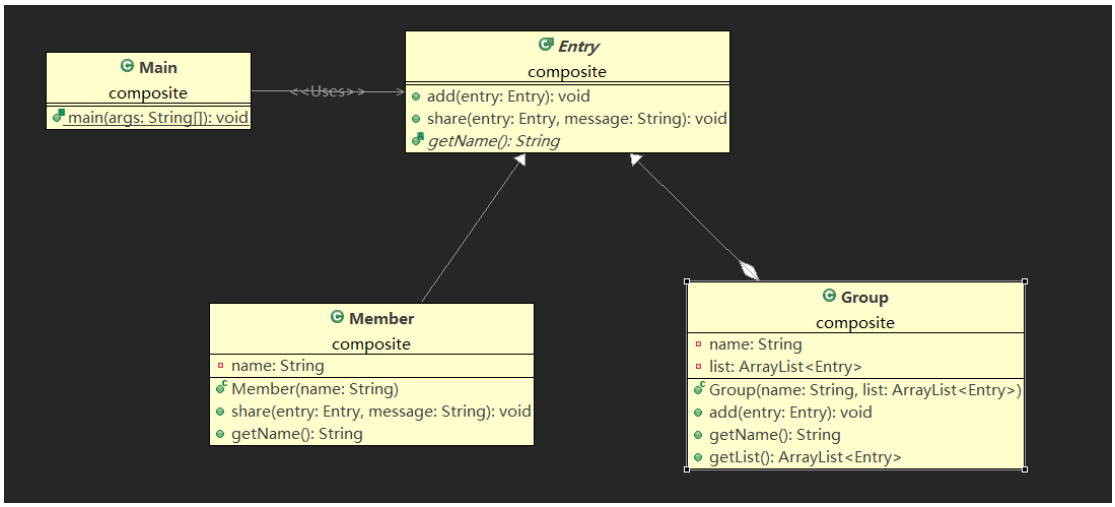


代码实现：

```
public interface Strategy {

    public void migration();

}

public class PreCopyStrategy implements Strategy{

    @Override
    public void migration() {
        System.out.println("Implementation of the PreCopy Migration Algorithm");

    }

}
```

```
public class PostCopyStrategy implements Strategy{

    @Override
    public void migration() {
        System.out.println("Implementation of the PostCopy Migration Algorithm");

    }

}

public class CRRTMotionStrategy implements Strategy{

    @Override
    public void migration() {
        System.out.println("Implementation of the CR/RTMotion Migration
Algorithm");

    }

}

public class CloudComputingPlatform {

    private Strategy strategy;

    public CloudComputingPlatform(Strategy strategy) {
        this.strategy = strategy;
    }

    public void vmMigration() {
        strategy.migration();
    }

}

public class Main {

    public static void main(String[] args) {
        CloudComputingPlatform p1 = new CloudComputingPlatform(new
PreCopyStrategy());
        CloudComputingPlatform p2 = new CloudComputingPlatform(new
PostCopyStrategy());
        CloudComputingPlatform p3 = new CloudComputingPlatform(new
CRRTMotionStrategy());

        p1.vmMigration();
        p2.vmMigration();
        p3.vmMigration();
    }

}
```

3. 组合模式



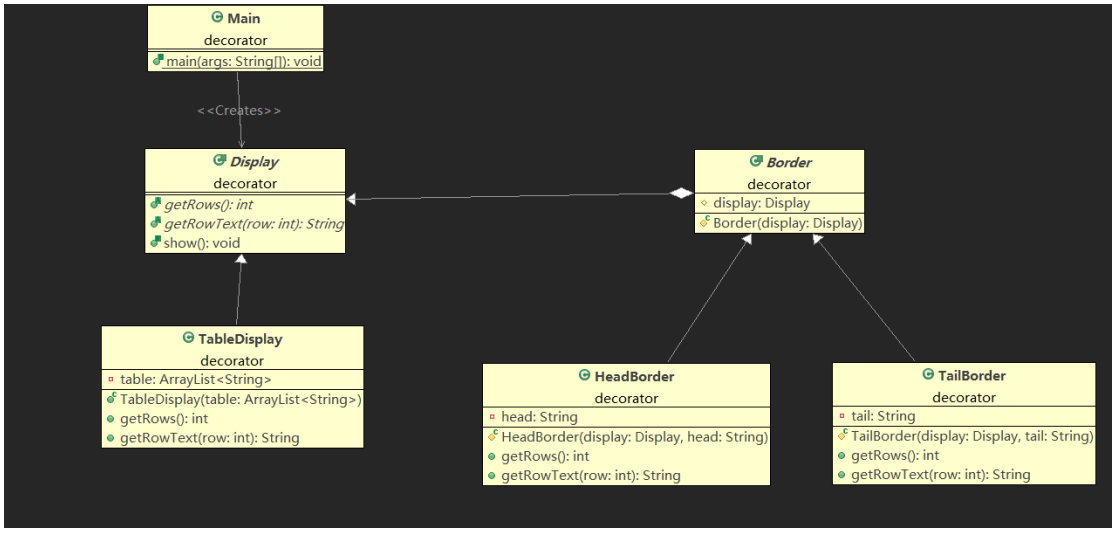
代码实现：

```
public abstract class Entry {

    public void add(Entry entry) throws MemberAddException {
        throw new MemberAddException();
    }

    public void share(Entry entry, String message) throws GroupShareException {
        throw new GroupShareException();
    }

    public abstract String getName();

}

public class Group extends Entry{

    private String name;
    private ArrayList<Entry> list;

    public Group(String name, ArrayList<Entry> list) {
        this.name = name;
        this.list = list;
    }

    @Override
    public void add(Entry entry) {
        if (entry instanceof Member) {
            list.add(entry);
        }
        else {
            Group group = (Group)entry;
            ArrayList<Entry> list = group.getList();
            for(Entry innerEntry: list) {
                this.add(innerEntry);
            }
        }
    }
```

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

    public ArrayList<Entry> getList() {
        return list;
    }

}

public class Member extends Entry{

    private String name;

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

    @Override
    public void share(Entry entry, String message){
        if (entry instanceof Member) {
            System.out.println(name + " share " + message + " with " +
entry.getName());
        }
        else {
            Group group = (Group)entry;
            ArrayList<Entry> list = group.getList();
            for(Entry innerEntry: list) {
                this.share(innerEntry, message);
            }
        }
    }


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

}

public class GroupShareException extends Exception {

}

public class MemberAddException extends Exception {

}

public class Main {

    public static void main(String[] args) {
        Member tom = new Member("Tom");
        Member jerry = new Member("Jerry");
        Member nancy = new Member("Nancy");
        Member jack = new Member("Jack");
```

```
        ArrayList<Entry> list1 = new ArrayList<>();
        list1.add(jerry);
        list1.add(nancy);

        ArrayList<Entry> list2 = new ArrayList<>();
        list2.add(jack);
        list2.add(tom);

        Group g1 = new Group("G1", list1);
        Group g2 = new Group("G2", list2);
        g1.add(g2);

        tom.share(g1, "hello");
        tom.share(nancy, "hello nancy");

    }
}
```

4. 装饰模式



代码实现：

```
public abstract class Display {

    public abstract int getRows();
    public abstract String getRowText(int row);

    public final void show() {
        for (int i = 0; i < getRows(); i++) {
            System.out.println(getRowText(i));
        }
    }

}

public class TableDisplay extends Display{

    private ArrayList<String> table;
```

```
    public TableDisplay(ArrayList<String> table) {
        this.table = table;
    }


    @Override
    public int getRows() {
        return table.size();
    }

    @Override
    public String getRowText(int row) {
        return table.get(row);
    }

}

public abstract class Border extends Display{

    protected Display display;
    protected Border(Display display) {
        this.display = display;
    }

}

public class HeadBorder extends Border{

    private String head;

    protected HeadBorder(Display display, String head) {
        super(display);
        this.head = head;
    }

    @Override
    public int getRows() {
        return display.getRows() + 1;
    }

    @Override
    public String getRowText(int row) {
        if (row == 0) {
            return head;
        }
        else {
            return display.getRowText(row - 1);
        }
    }

}

public class TailBorder extends Border{

    private String tail;

    protected TailBorder(Display display, String tail) {
        super(display);
```

```
        this.tail = tail;
    }

    @Override
    public int getRows() {
        return display.getRows() + 1;
    }

    @Override
    public String getRowText(int row) {
        if (row == display.getRows()) {
            return tail;
        }
        else {
            return display.getRowText(row);
        }
    }

}

public class Main {


    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("A:123");
        list.add("B:456");
        Display d1 = new TableDisplay(list);
        d1.show();

        Display d2 = new TailBorder(d1, "tail");
        d2.show();

        Display d3 = new HeadBorder(new HeadBorder(new TailBorder(d1, "Tail"),
"Head2"), "Head1");
        d3.show();
    }

}
```

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，我加强了对桥接模式、策略模式、组合模式，装饰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。

组合模式中，遇到的问题：Group 和 Member 类的 add 和 share 方法中要判断类型来实现不同的功能。
解决办法 :可以使用 instanceof 检查目标对象的类型，以避免 ClassCastException 异常。