设计模式实验（3）

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. 简单工厂模式

简单工厂模式使用简单工厂模式设计一个可以创建不同几何形状（Shape）（例如圆形（Circle）、矩形（Rectangle）和三角形（Triangle）等）的绘图工具类，每个几何图形均具有绘制方法 draw()和擦除方法 erase()，要求在绘制不支持的几何图形时，抛出一个UnsupportedShapeException 异常。绘制类图并编程模拟实现。

2. 建造者模式

在某赛车游戏中，赛车包括方程式赛车、场地越野赛车、运动汽车、卡车等类型，不同类型的赛车的车身、发动机、轮胎、变速箱等部件有所区别。玩家可以自行选择赛车类型，系统将根据玩家的选择创建出一辆完整的赛车。现采用建造者模式实现赛车的构建，绘制对应的类图并编程模拟实现

3. 抽象工厂模式

某系统为了改进数据库操作的性能，用户可以自定义数据库连接对象 Connection 和语句对象 Statement ，针对不同类型的数据库提供不同的连接对象和语句对象，例如提供Oracle 或 MySQL 专用连接类和语句类，而且用户可以通过配置文件等方式根据实际需要动态更换系统数据库。使用抽象工厂模式设计该系统，绘制对应的类图并编程模拟实现

4. 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。

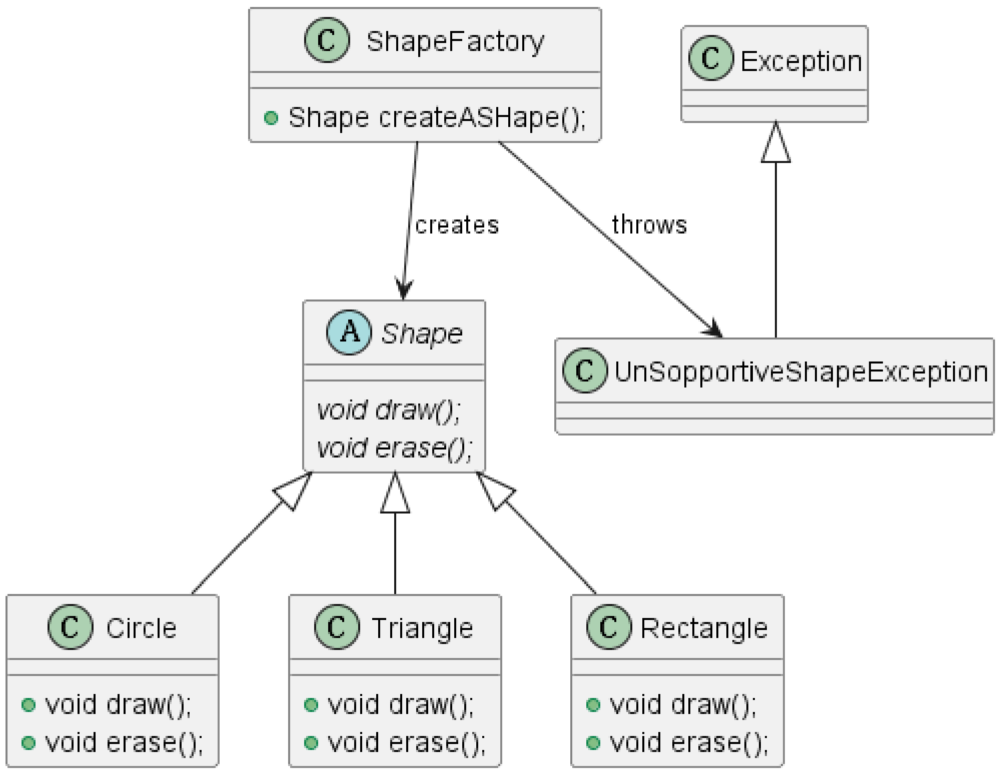
5. 策略模式

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 PreCopy （预拷贝）算法、PostCopy （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。

三、实验结果

1.简单工厂模式

类图



实现代码

- Circle.java

```java
public class Circle extends Shape{

    @Override
    public void draw() {
        System.out.println("Draw a new "+this.getClass().getName());
    }

    @Override
    public void erase() {
        System.out.println("Erase the "+this.getClass().getName());
    }
}
```

- Rectangle.java

```java
public class Rectangle extends Shape{
    @Override
    public void draw() {
        System.out.println("Draw a new "+this.getClass().getName());
    }

    @Override
    public void erase() {
        System.out.println("Erase the "+this.getClass().getName());
    }
}
```

- Shape.java

```java
public abstract class Shape {
    abstract void draw();
    abstract void erase();
}
```

- ShapeFactory.java

```java

public class ShapeFactory {

    public Shape createAShape(String shape) throws UnsupportiveShapeException{
        if(shape.equals("Circle")){
            return new Circle();
        } else if(shape.equals("Rectangle")){
            return new Rectangle();
        } else if(shape.equals("Triangle")){
            return new Triangle();
        } else {
            throw new UnsupportiveShapeException();
        }
    }
}
```

- Test_3_1.java

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

public class Test_3_1 {
    public static void main(String[] args) {
        List<String> shapes= new ArrayList<String>();
        shapes.add("Circle");
        shapes.add("Triangle");
        shapes.add("Rectangle");
        shapes.add("Trapezium");
        ShapeFactory shapeFactory =new ShapeFactory();
        Shape s1=null;
        for (String s:shapes) {
            try {
                s1 = shapeFactory.createAShape(s);
                s1.draw();
                s1.erase();
            } catch (UnsupportiveShapeException e) {
                e.printStackTrace();
            }
            s1 = null;
        }
    }
}
```

- Triangle.java

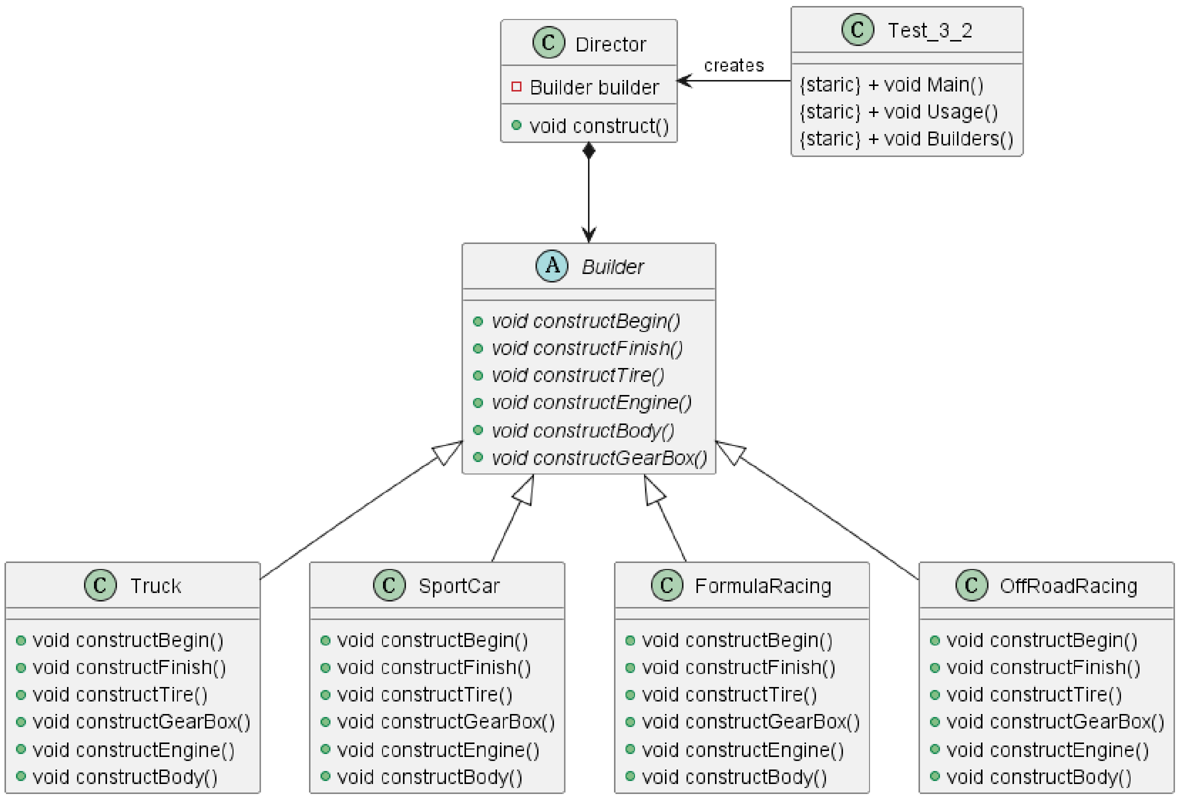
```
public class Triangle extends Shape{
    @Override
    public void draw() {
        System.out.println("Draw a new "+this.getClass().getName());
    }

    @Override
    public void erase() {
        System.out.println("Erase the "+this.getClass().getName());
    }
}
```

- UnsupportiveShapeException.java

```
public class UnsupportiveShapeException extends Exception{

}
```

2.建造者模式

类图



实现代码

- Builder.java

```
public abstract class Builder {
    abstract void constructBody();
    abstract void constructEngine();
    abstract void constructTire();
    abstract void constructGearBox();
    abstract void constructFinish();
    abstract void constrcutBegin();
}
```

- Director.java

```
public class Director {
    private Builder builder;
    public Director(Builder builder){
        this.builder=builder;
    }

    public void construct(){
        builder.constrcutBegin();
        builder.constructBody();
        builder.constructEngine();
        builder.constructTire();
        builder.constructGearBox();
        builder.constructFinish();
    }

}
```

- FormulaRacing.java

```
public class FormulaRacing extends Builder{
    @Override
    void constructBody() {
        System.out.println("Construct the Body of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructEngine() {
        System.out.println("Construct the Engine of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructTire() {
        System.out.println("Construct the Tire of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructGearBox() {
        System.out.println("Construct the GearBox of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructFinish() {
        System.out.println("\n========== Constructing 
"+this.getClass().getName()+" finished ==========\n");
    }

    @Override
    void constrcutBegin() {
        System.out.println("\n========== Begin to construct the "
+this.getClass().getName() + " ==========\n");
    }
}
```

- OffRoadRacing.java

```
public class OffRoadRacing extends Builder{
    @Override
    void constructBody() {
        System.out.println("Construct the Body of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructEngine() {
        System.out.println("Construct the Engine of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructTire() {
        System.out.println("Construct the Tire of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructGearBox() {
        System.out.println("Construct the GearBox of Your 
"+this.getClass().getName()) ;
    }

    @Override
    void constructFinish() {
        System.out.println("\n========== Constructing 
"+this.getClass().getName()+" finished ==========\n");
    }

    @Override
    void constrcutBegin() {
        System.out.println("\n========== Begin to construct the "
+this.getClass().getName() + " ==========\n");
    }

}
```

- SportCar.java

```
public class SportCar extends Builder{
    @Override
    void constructBody() {
        System.out.println("Construct the Body of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructEngine() {
        System.out.println("Construct the Engine of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructTire() {
        System.out.println("Construct the Tire of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructGearBox() {
        System.out.println("Construct the GearBox of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructFinish() {
        System.out.println("\n========== Constructing
"+this.getClass().getName()+" finished ==========\n");
    }

    @Override
    void constrcutBegin() {
        System.out.println("\n========== Begin to construct the "
+this.getClass().getName() + " ==========\n");
    }
}
```

- Test_3_2.java

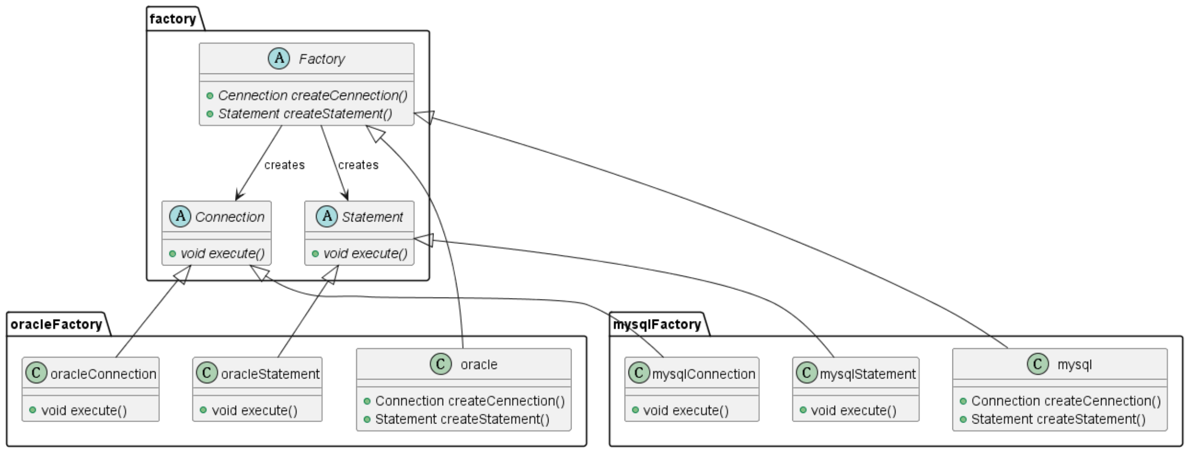
```
import java.util.ArrayList;
import java.util.List;

public class Test_3_2 {

    static List<String> builders = new ArrayList<String>();

    public static void main(String[] args) {
        Builders();
        if(args.length!=1){
            Usage();
            System.exit(0);
        }
        Director director=null;
        for (String s:builders){
            if (s.equals(args[0])){
                try {
                    Builder builder = (Builder) Class.forName(s).newInstance();
                    director=new Director(builder);
                    director.construct();
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                } catch (InstantiationException e) {
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                }
                System.exit(0);
            }
        }
        Usage();
    }

    static void Usage(){
        System.out.println("usage:");
        for (String s:builders){
            System.out.println("Java Test_3_2 "+s);
        }

    }

    static void Builders(){
        builders.add(Truck.class.getName());
        builders.add(SportCar.class.getName());
        builders.add(FormulaRacing.class.getName());
        builders.add(OffRoadRacing.class.getName());
    }
}
```

- Truck.java

```
public class Truck extends Builder{
    @Override
    void constructBody() {
        System.out.println("Construct the Body of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructEngine() {
        System.out.println("Construct the Engine of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructTire() {
        System.out.println("Construct the Tire of Your
"+this.getClass().getName()) ;
    }

    @Override
    void constructGearBox() {
        System.out.println("Construct the GearBox of Your
"+this.getClass().getName()) ;
    }
    @Override
    void constructFinish() {
        System.out.println("\n========== Constructing
"+this.getClass().getName()+" finished ==========\n");
    }

    @Override
    void constrcutBegin() {
        System.out.println("\n========== Begin to construct the "
+this.getClass().getName() + " ==========\n");
    }
}
```

3.抽象工厂模式

类图



实现代码

- Connection3_3.java

```
public abstract class Connection3_3 {
    public abstract void execute();
}
```

- Factory.java

```
public abstract class Factory {
    public abstract Connection3_3 createConnetion();
    public abstract Statement createStatement();
}
```

- mysql.java

```java
public class mysql extends Factory{
    @Override
    public Connection3_3 createConnetion() {
        return new mysqlConnection();
    }

    @Override
    public Statement createStatement() {
        return new mysqlStatement();
    }
}
```

- mysqlConnection.java

```java
public class mysqlConnection extends Connection3_3{
    @Override
    public void execute() {
        System.out.println("Using MySql Connection");
    }
}
```

- mysqlStatement.java

```java
public class mysqlStatement extends Statement{
    @Override
    public void execute() {
        System.out.println("Using MySql Statamemnts");
    }
}
```

- oracle.java

```java
public class oracle extends Factory{
    @Override
    public Connection3_3 createConnetion() {
        return new oracleConnection();
    }

    @Override
    public Statement createStatement() {
        return new oracleStatement();
    }
}
```

- oracleConnection.java

```java
public class oracleConnection extends Connection3_3{
    @Override
    public void execute() {
        System.out.println("Using Oracle Connection");
    }
}
```

- oracleStatement.java

```
public class oracleStatement extends Statement {
    @Override
    public void execute() {
        System.out.println("Using Oracle Statamemnts");
    }
}
```

- Statement.java

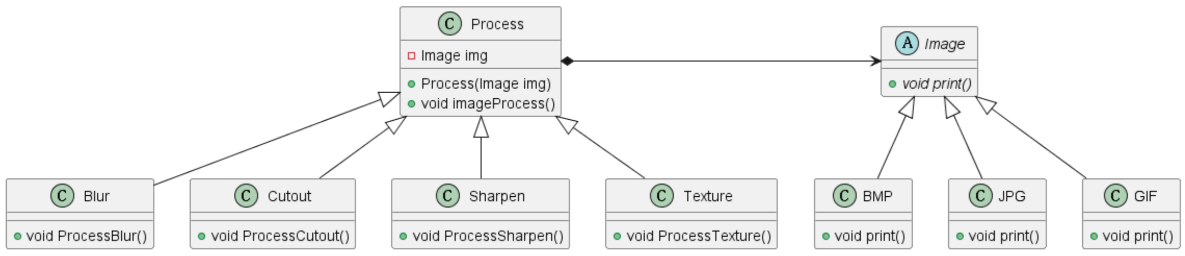
```
public abstract class Statement {
    public abstract void execute();
}
```

- Test_3_3.java

```
public class Test_3_3 {
    public static void main(String[] args) {
        if(args.length!=1){
            Usage();
            System.exit(0);
        }
        try {
            Factory db = (Factory) Class.forName(args[0]).newInstance();
            Connection3_3 con = db.createConnetion();
            Statement sta = db.createStatement();
            con.execute();
            sta.execute();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    static void Usage(){
        System.out.println("Usage:");
        System.out.println("Java Test_3_3 oracle");
        System.out.println("Java Test_3_3 mysql");
    }

}
```

4.桥接模式

类图



实现代码

- Blur.java

```java
public class Blur extends Process{
    public Blur(Image image) {
        super(image);
    }

    public void processBlur(){
        System.out.println("==== "+ this.getClass().getName() + " =====");
        imageProcess();
        System.out.println("==== "+ this.getClass().getName() + " =====");
    }
}
```

- BMP.java

```java
public class BMP extends Image{

    @Override
    void print() {
        System.out.println("this is a(n) "+this.getClass().getName() + " 
Image");
    }
}
```

- Cutout.java

```java
public class Cutout extends Process{
    public Cutout(Image image) {
        super(image);
    }

    public void processCutout(){
        System.out.println("==== "+ this.getClass().getName() + " =====");
        imageProcess();
        System.out.println("==== "+ this.getClass().getName() + " =====");
    }
}
```

- GIF.java

```java
public class GIF extends Image{
    @Override
    void print() {
        System.out.println("this is a(n) "+this.getClass().getName() + " 
Image");
    }
}
```

- Image.java

```
public abstract class Image {
    abstract void print();
}
```

- JPG.java

```
public class JPG extends Image{
    @Override
    void print() {
        System.out.println("this is a(n) "+this.getClass().getName() + " 
Image");
    }
}
```

- Process.java

```
public class Process {
    private Image image;
    public Process(Image image){
        this.image=image;
    }
    public void imageProcess(){
        image.print();
    }
}
```

- Sharpen.java

```
public class Sharpen extends Process{
    public Sharpen(Image image) {
        super(image);
    }

    public void processSharpen(){
        System.out.println("==== "+ this.getClass().getName() + " =====");
        imageProcess();
        System.out.println("==== "+ this.getClass().getName() + " =====");
    }
}
```

- Test_3_4.java

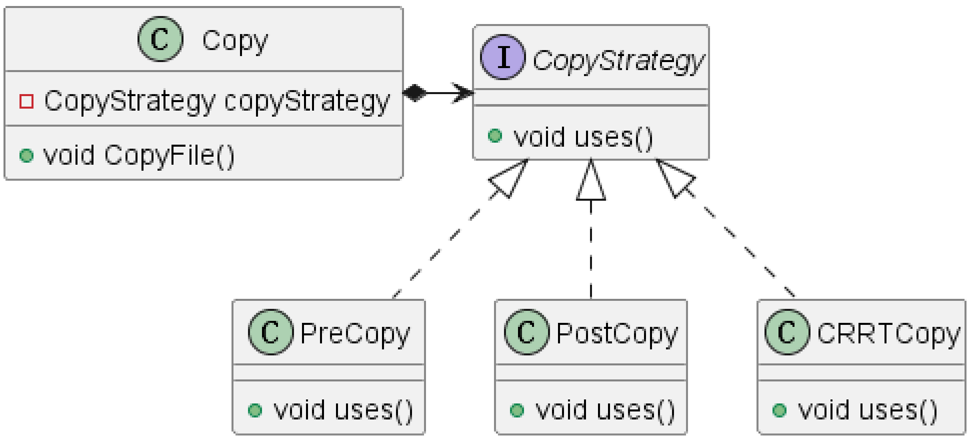
```
import java.util.ArrayList;
import java.util.List;

public class Test_3_4 {
    static List<Image> imgs;
    public static void main(String[] args) {
        Addimgs();
        Blur p1 = null;
        // Blur
        for(Image img:imgs){
            p1 = new Blur(img);
            p1.processBlur();
            System.out.println();
        }

        // Cutout
        Cutout p2 = null;
        for(Image img:imgs){
            p2 = new Cutout(img);
            p2.processCutout();
            System.out.println();
        }

        // Sharpen
        Sharpen p3 = null;
        for(Image img:imgs){
            p3 = new Sharpen(img);
            p3.processSharpen();
            System.out.println();
        }

        // Texture
        Texture p4;
        for(Image img:imgs){
            p4 = new Texture(img);
            p4.processTexture();
            System.out.println();
        }
    }

    static void Addimgs(){
        imgs = new ArrayList<Image>();
        imgs.add(new BMP());
        imgs.add(new GIF());
        imgs.add(new JPG());
    }
}
```

- Texture.java

```
public class Texture extends Process{

    public Texture(Image image) {
        super(image);
    }

    public void processTexture(){
        System.out.println("==== "+ this.getClass().getName() + " =====");
        imageProcess();
        System.out.println("==== "+ this.getClass().getName() + " =====");
    }
}
```

5.策略模式

类图



实现代码

- Copy.java

```java
public class Copy {
    private CopyAlgorithm copyAlgorithm;
    public Copy(CopyAlgorithm copyAlgorithm){
        this.copyAlgorithm=copyAlgorithm;
    }
    public void CopyFile(){
        copyAlgorithm.uses();
    }
}
```

- CopyAlgorithm.java

```java
public interface CopyAlgorithm {
    public void uses();
}
```

- CRRTMotion.java

```java
public class CRRTMotion implements CopyAlgorithm{
    @Override
    public void uses() {
        System.out.println("Using "+this.getClass().getName()+" Algorithm");
    }
}
```

- PostCopy.java

```java
public class PostCopy implements CopyAlgorithm{
    @Override
    public void uses() {
        System.out.println("Using "+this.getClass().getName()+" Algorithm");
    }
}
```

- PreCopy.java

```java
public class PreCopy implements CopyAlgorithm{
    @Override
    public void uses() {
        System.out.println("Using "+this.getClass().getName()+" Algorithm");
    }
}
```

- Test_3_5.java

```java
public class Test_3_5 {
    public static void main(String[] args) {
        Copy c1 = new Copy(new PreCopy());
        Copy c2 = new Copy(new PostCopy());
        Copy c3 = new Copy(new CRRTMotion());
        c1.CopyFile();
        c2.CopyFile();
        c3.CopyFile();
    }
}
```

五、实验小结

本次实验学会了简单工厂模式，建造者模式，抽象工厂模式,桥接模式和策略模式，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