Decorator Design Pattern

```
interface Shape{
    void draw();
}

class Rectange implements Shape{

    public void draw(){
        System.out.println("Drawing Rectange...");
    }
}

class ShapeDecorator implements Shape{

    protected Shape shape;

    ShapeDecorator(Shape s){
        this.shape = s;
    }

    public void draw(){
        shape.draw();
    }

}

class RedColorRectangeDecorator extends ShapeDecorator{

    RedColorRectangeDecorator(Shape s){
        super(s);
    }

    public void draw(){
        shape.draw();
        setRedColor();
    }

    private void setRedColor(){
        System.out.println("Set color of shape to red by changing some property")
;
    }
```

```
}

public class Decorator{

    public static void main(String[] args){

        Shape shape = new Rectange();
        shape = new RedColorRectangeDecorator(shape);

        shape.draw();


    }

}
```

Composite Design Pattern

```
import java.util.List;
import java.util.*;

class Employee{

    private int id;
    private String name;
    protected List<Employee> subOrdinates;

    Employee(int id, String name){
        this.id = id;
        this.name = name;
        subOrdinates = new ArrayList<Employee>();
    }

    public void addSubOrdinates(Employee em){
        subOrdinates.add(em);
    }

    public void printSubOrdinates(){
        subOrdinates.stream().forEach(e -> System.out.println(e.name));
    }
}


public class Composite{
```

```
    public static void main(String[] args){

        Employee manager = new Employee(1, "Manager");
        Employee lead = new Employee(2, "Lead");
        Employee sse = new Employee(3, "sse");
        Employee specialist = new Employee(4, "Specialist");
        manager.addSubOrdinates(lead);
        manager.addSubOrdinates(specialist);

        lead.addSubOrdinates(sse);

        manager.printSubOrdinates();
        lead.printSubOrdinates();
    }
}
```

Observer Pattern

```
import java.util.*;

interface Subscriber{
    void update();
}

interface Publisher{
    void publish();
    void addSubscriber(Subscriber subscriber);
}

class PrimePublisher implements Publisher{
    String name;
    List<Subscriber> subscribers;

    PrimePublisher(String name){
        this.name = name;
        subscribers = new ArrayList<>();
    }

    public void addSubscriber(Subscriber subscriber){
        subscribers.add(subscriber);
    }
```

```
    public void publish(){
        subscribers.stream().forEach(s -> s.update());
    }
}



class FirstSub implements Subscriber{

    @Override
    public void update() {

        System.out.println("Subscriber FirstSub update called");

    }


}

class SecondSub implements Subscriber{

    @Override
    public void update() {

        System.out.println("Subscriber SecondSub update called");

    }


}


public class Observer{

    public static void main(String[] args){
         Publisher publisher = new PrimePublisher("Prime");
         publisher.addSubscriber(new FirstSub());
         publisher.addSubscriber(new SecondSub());

         publisher.publish();
    }
}
```

Iterator Design Pattern

```java
interface Iterator{
    boolean hasNext();
    Object next();
}

class NameList{

    String[] names = {"Viraj", "MS", "SH"};
    public Iterator getIterator(){
        return new NameListIterator();
    }
    private class NameListIterator implements Iterator{

        private int index;
        @Override
        public boolean hasNext() {

            if(index < names.length)
            return true;
            else
            return false;
        }

        @Override
        public Object next() {

            if(this.hasNext())
            return names[index++];
            else
            return null;
        }


    }


}


public class IteratorDesign{

    public static void main(String[] args){
        NameList nameList = new NameList();
        Iterator it = nameList.getIterator();
```

```
        while(it.hasNext()){
             System.out.println(it.next());
        }

    }
}
```

Template Method

```
abstract class Algorithm{

public abstract void stepOne();
public abstract void stepTwo();
    public void stepThree(){
        System.out.println("This is step 3");
    }
    public void performAllSteps(){
       stepOne();
       stepTwo();
       stepThree();
    }
}

class FirstAlgo extends Algorithm{

    @Override
    public void stepOne() {
        // TODO Auto-generated method stub

        System.out.println("First algo step one");
    }

    @Override
    public void stepTwo() {
        // TODO Auto-generated method stub
        System.out.println("First algo step two");
    }

}

class SecondAlgo extends Algorithm{
    @Override
    public void stepOne() {
        // TODO Auto-generated method stub
```

```
        System.out.println("Second algo step one");
    }

    @Override
    public void stepTwo() {
        // TODO Auto-generated method stub
        System.out.println("Second algo step two");
    }

    public void stepThree(){
        System.out.println("This is step 3 modified");
    }



}


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

          Algorithm algorithm = new FirstAlgo();
          algorithm.performAllSteps();

          Algorithm algorithm2 = new SecondAlgo();
          algorithm2.performAllSteps();

    }
}
```