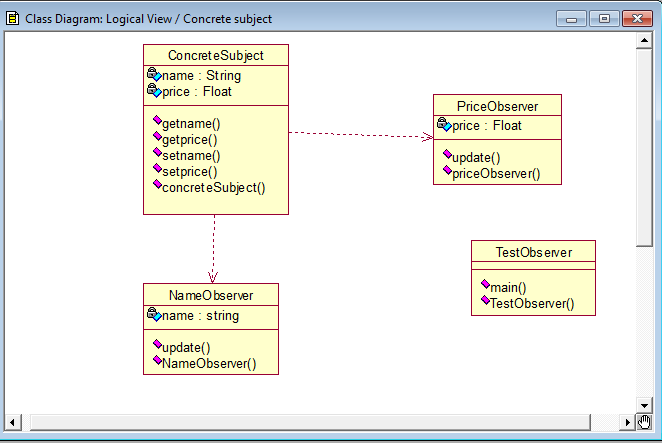
# Contents

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Pattern Name** | **Page No** |
| **1** | **Publisher-Subscriber** | **2** |
| **2** | **Command Processor** | **10** |
| **3** | **Client-Dispatcher** | **18** |
| **4** | **polymorphism** | **25** |
| **5** | **Whole-Part** | **30** |
| **6** | **Forward Receiver** | **35** |

# 1.Publisher-Subscriber pattern

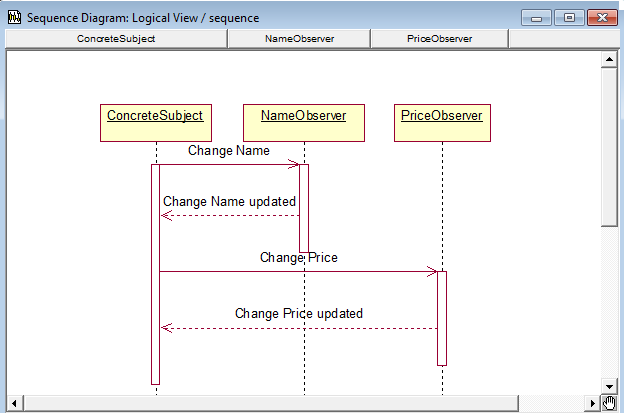
### 1use Use Case Diagram:

Class Diagram:

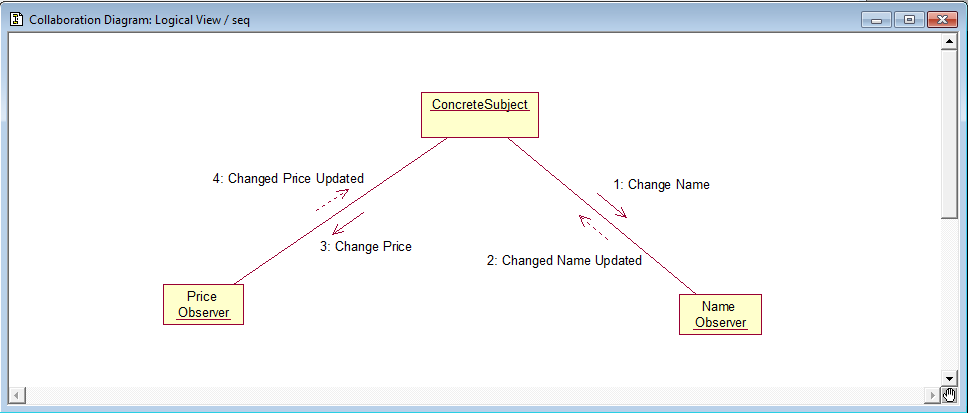


## Dynamics:

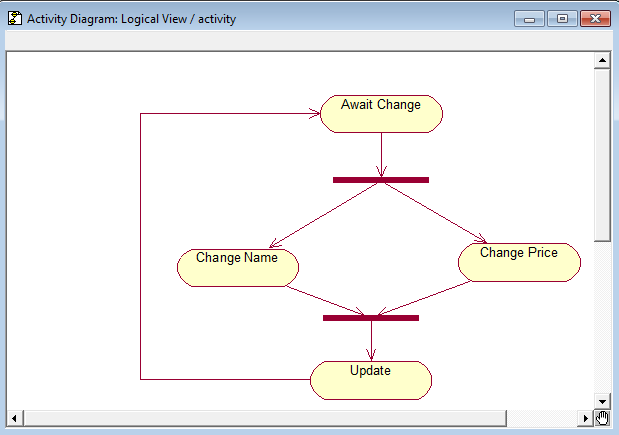
### Sequence diagram:



Collaboration diagram:



* Activity diagram:



## Implementation:

### ConcreteSubject.java

import java.lang.\*; import java.util.\*;

public class ConcreteSubject extends Observable

{

private String Name; private Float Price;

public ConcreteSubject(String Name,float Price)

{

this.Name=Name; this.Price=Price;

System.out.println("\n Concrete Subject Created"+Name+"at"+Price);

}

public String getName()

{

return Name;

}

public float getPrice()

{

return Price;

}

public void setName(String Name)

this.Name=Name; setChanged(); notifyObservers(Name);

}

public void setPrice(float Price)

{

this.Price=Price; setChanged();

notifyObservers(new Float(Price));

}

}

### NameObserver.java

import java.util.Observable; import java.util.Observer;

public class NameObserver implements Observer

{

private String Name; public NameObserver()

{

Name=null;

System.out.println("\n Name Observer Created! name is:"+Name);

}

public void update(Observable obj,Object arg)

if(arg instanceof String)

{

Name=(String)arg;

System.out.println("name observer:"+Name);

}

else

{

System.out.println("name observer:some other change to subject");

}

}

}

### PriceObserver.java

import java.util.Observable; import java.util.Observer;

public class PriceObserver implements Observer

{

private float Price; public PriceObserver()

{

Price=0;

System.out.println("\n PriceObserver Created!Price is:"+Price);

}

public void update(Observable obj,Object arg)

{

if(arg instanceof Float)

{

Price=((Float)arg).floatValue();

System.out.println("\n PriceObserver:Priece changed to"+Price);

}

else

{

System.out.println("\n PriceObserver:Priece changed to"+Price);

}

}

}

### TestObserver.java

import java.util.\*;

public class TestObserver

{

public static void main(String args[])

{

ConcreteSubject s=new ConcreteSubject("corn pops",1.29f); NameObserver nameobs=new NameObserver(); PriceObserver priceobs=new PriceObserver(); s.addObserver(nameobs);

s.addObserver(priceobs); s.setName("frostyed flakes");

s.setPrice(4.57f); s.setPrice(9.22f); s.setName("sugar crispies");

}

}

### Outputs:

C:\Java>java TestObserver

Concrete Subject Createdcorn popsat1.29 Name Observer Created! name is:null PriceObserver Created!Price is:0.0 PriceObserver:Priece changed to0.0 name observer:frostyed flakes PriceObserver:Priece changed to4.57

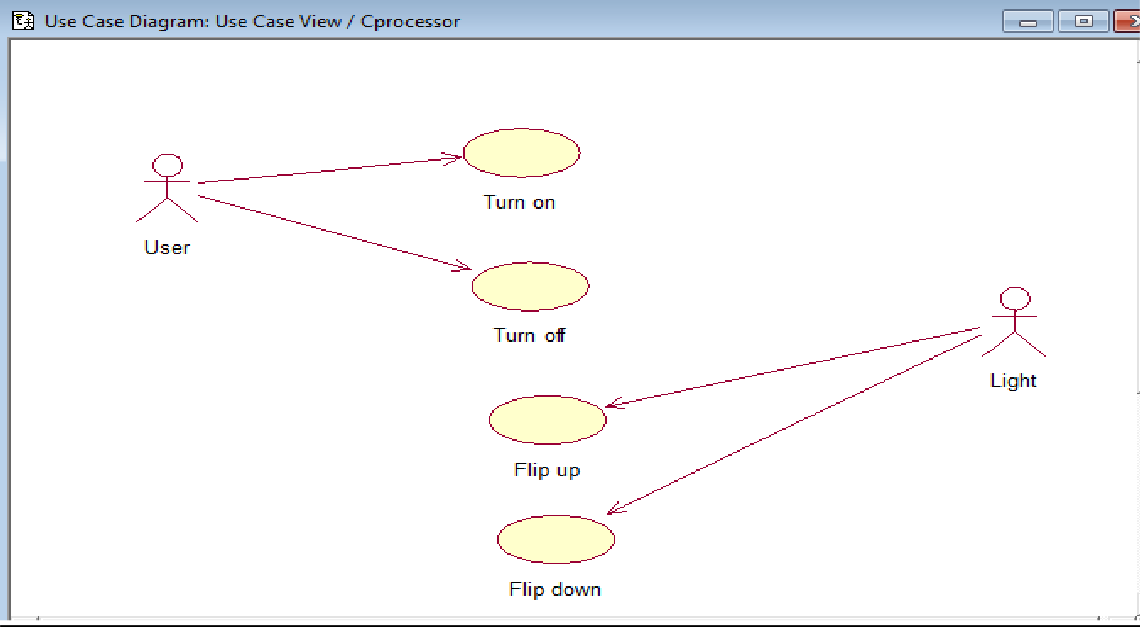
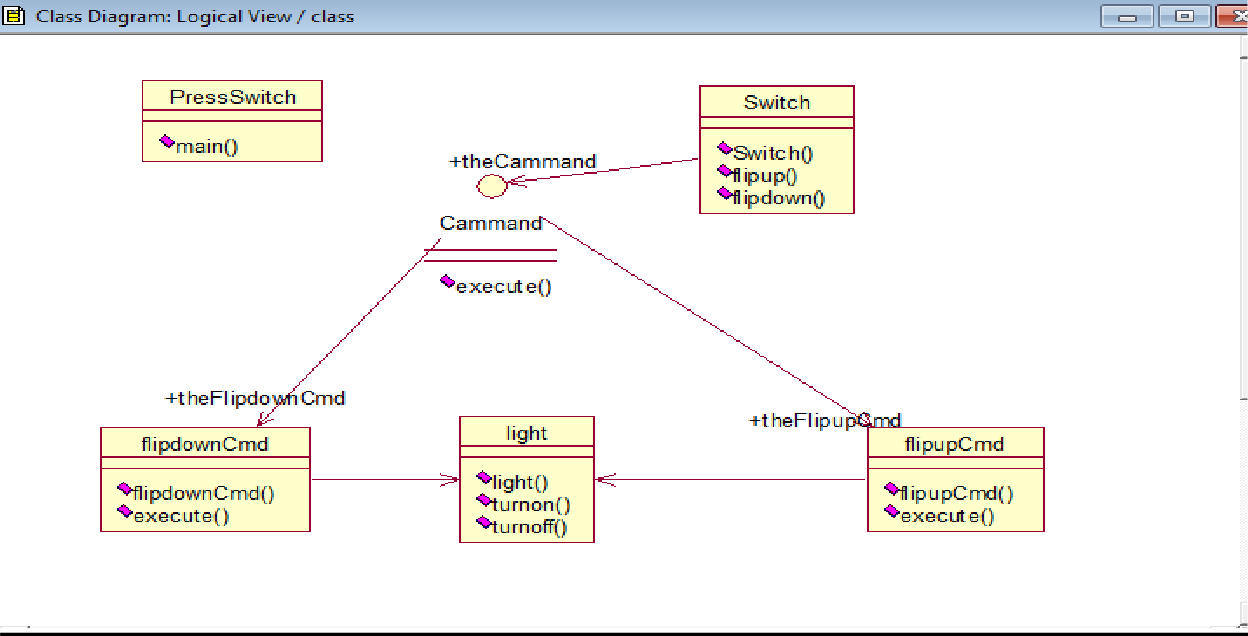
name observer:some other change to subject PriceObserver:Priece changed to9.22

name observer:some other change to subject PriceObserver:Priece changed to9.22

name observer:sugar crispies

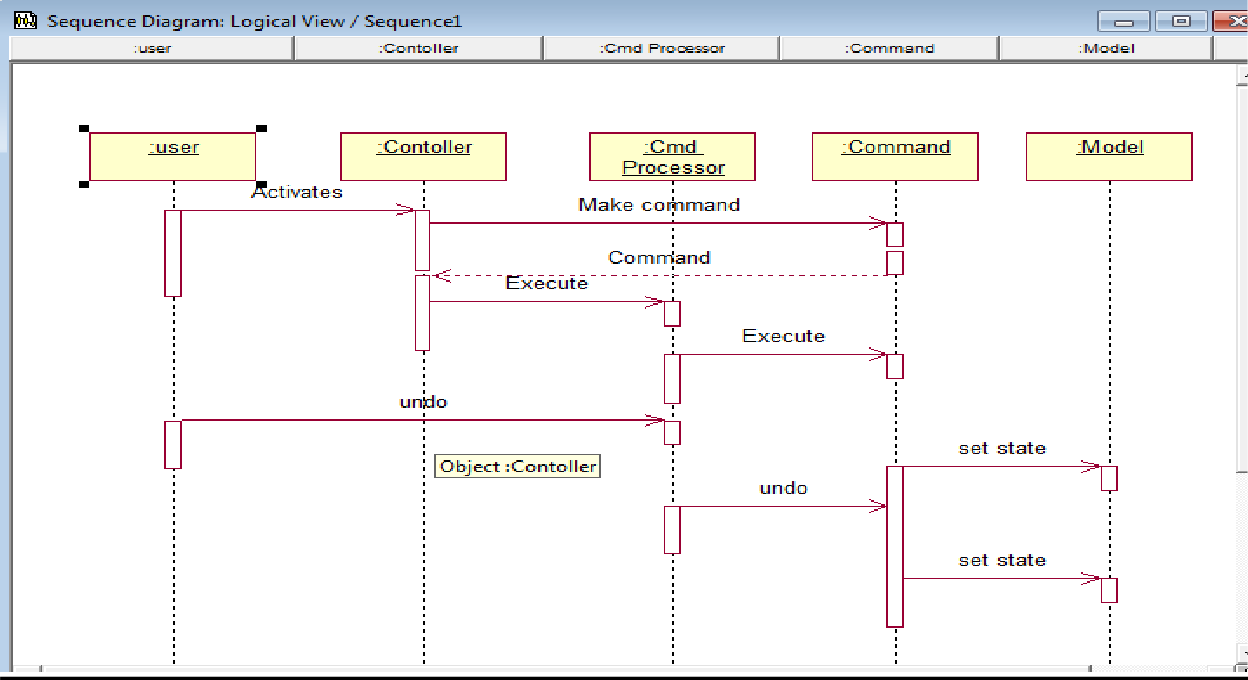
# Command Processor Pattern

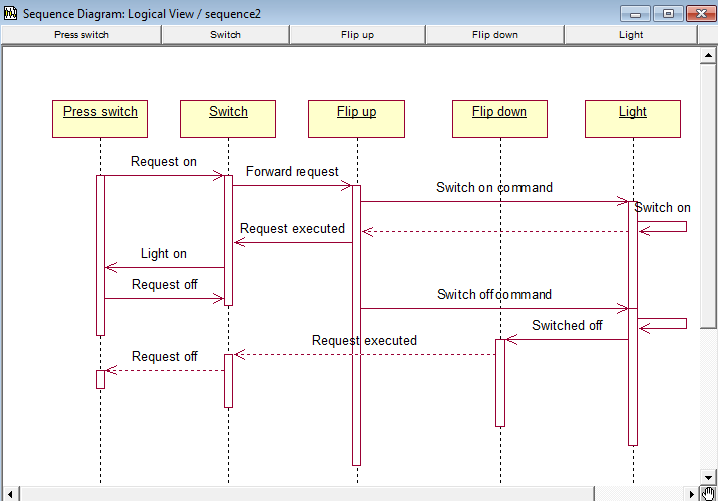
### Use Case Diagram:



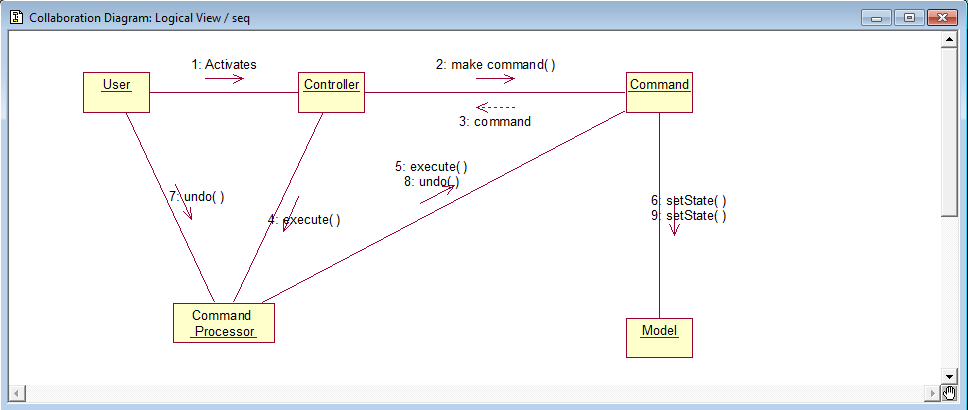
Class Diagram:

### Sequence diagram:

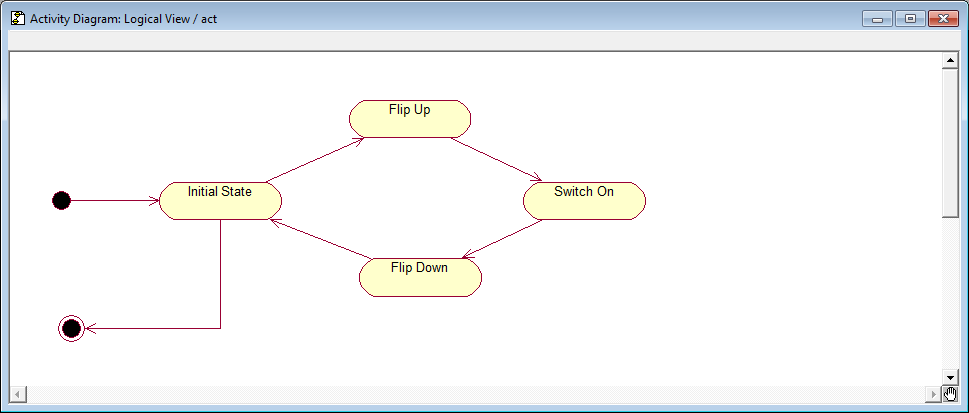




Collaboration diagram:



Activity diagram:



## Implementation:

### PrintSwitch.java

import java.io.\*; import java.io.\*; import java.lang.\*;

public class PressSwitch

{

public static void main(String args[])

{

light lamp=new light();

Cammand switchUp=new flipupCmd(lamp); Cammand switchDown=new flipdownCmd(lamp); Switch s=new Switch(switchUp,switchDown);

try

{

if(args[0].equalsIgnoreCase("ON")) s.flipUp();

else if(args[0].equalsIgnoreCase("OFF")) s.flipDown();

else

System.out.println("arguments required\n");

}

catch(Exception e)

{

System.out.println("Arguments required\n");

}

}

}

### Switch.java

public class Switch

{

private Cammand flipupCmd; private Cammand flipdownCmd;

public Switch(Cammand flipUpCmd,Cammand flipDownCmd)

{

this.flipupCmd=flipUpCmd;

this.flipdownCmd=flipDownCmd;

}

public void flipUp()

{

flipupCmd.execute();

}

public void flipDown()

{

flipdownCmd.execute();

}

}

### Light.java

public class light

{

public light()

{

}

public void turnOn()

{

System.out.println("\n\nthe light is on!\n");

}

public void turnOff()

{

System.out.println("\n\nthe light is off!\n");

}

}

### Cammand.java

import java.io.\*;

public interface Cammand

{

public void execute();

}

flipupCmd.java

public class flipupCmd implements Cammand

{

public light theLight;

public flipupCmd(light light)

{

this.theLight=light;

}

public void execute()

{

theLight.turnOn();

}

}

### flipdownCmd.java

public class flipdownCmd implements Cammand

{

public light theLight;

public flipdownCmd(light light)

{

this.theLight=light;

}

public void execute()

{

theLight.turnOff();

}

}

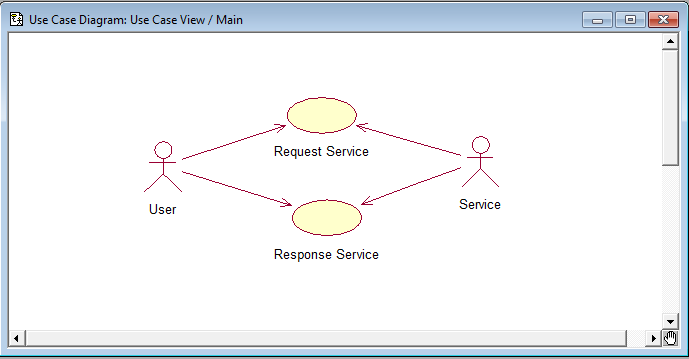
### Output:

C:\Java>java PressSwitch on the light is on!

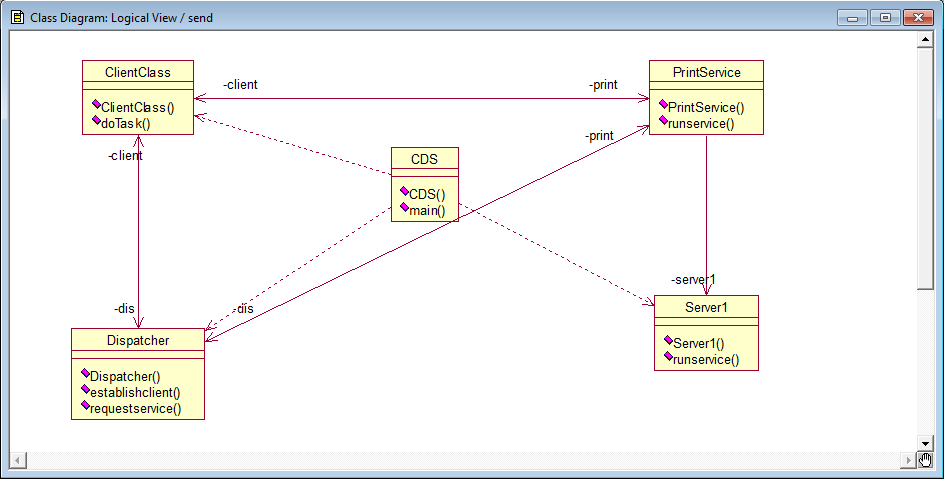
.

# Client-Dispatcher

### Use case Diagram:

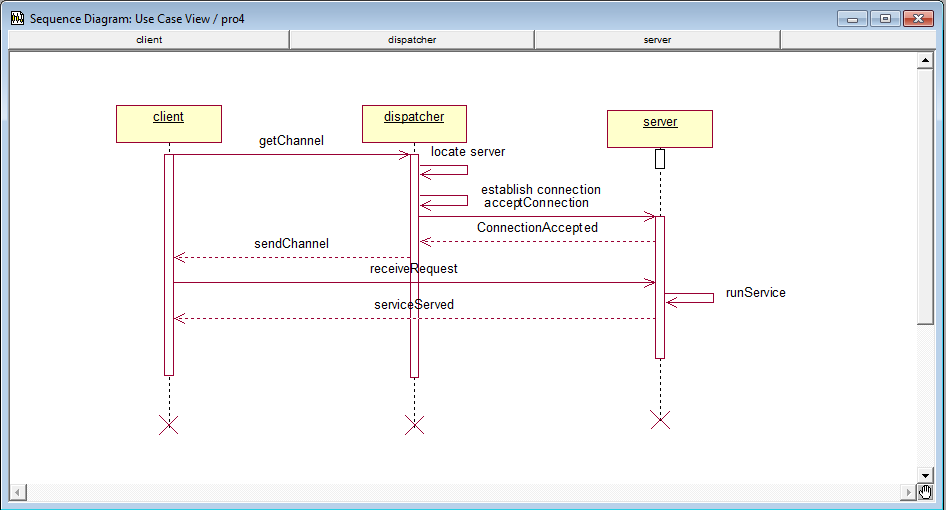


Class Diagram:



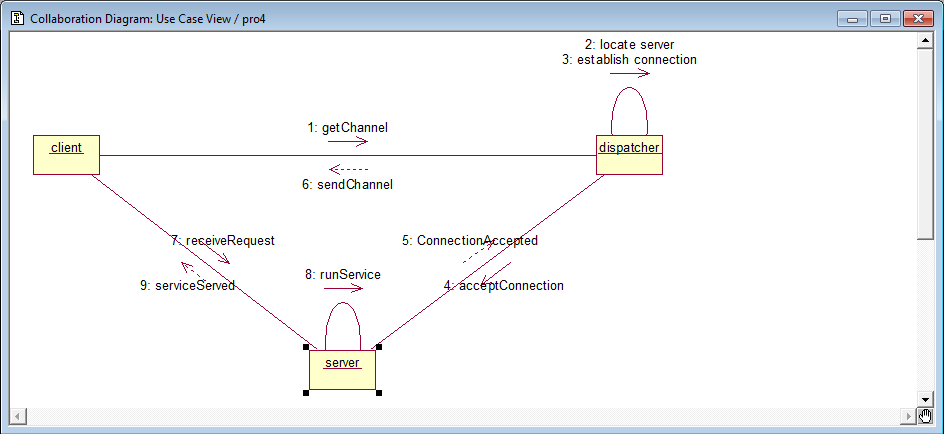
## Dynamics:

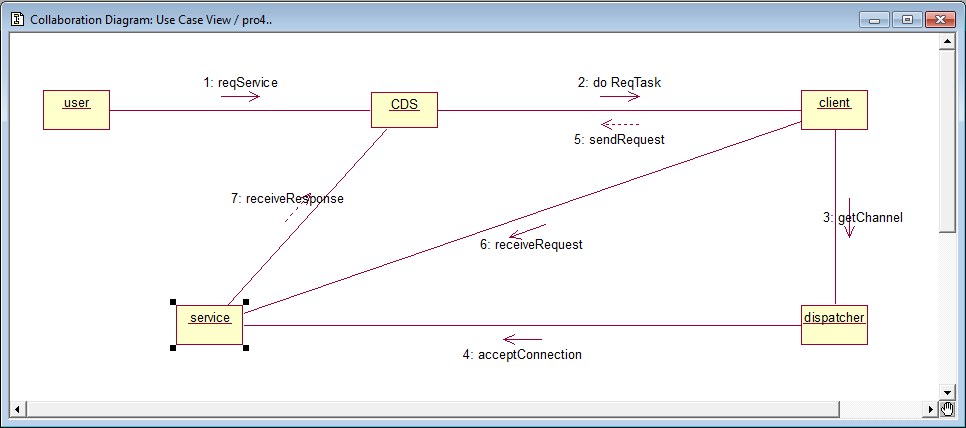
### Sequence diagram:

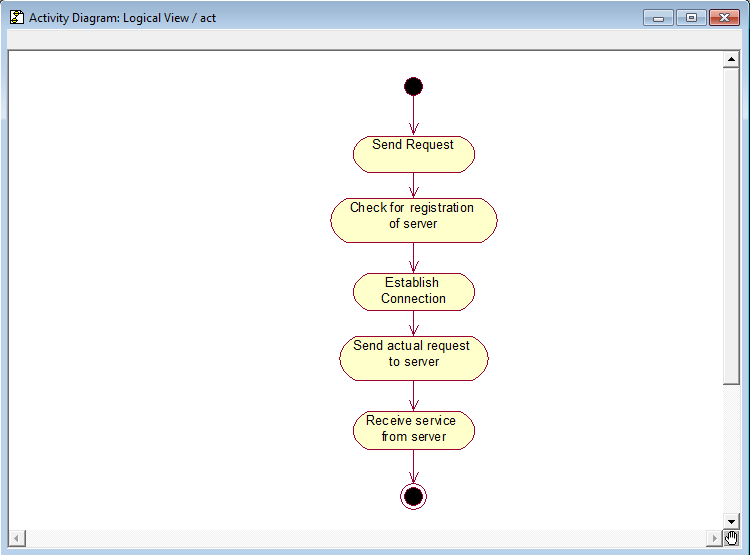


### ㇽ皼㎵琅숓

Collaboration diagram





**Activity Diagram**

## Implementation:

### Cds.java

import java.io.\*; import java.lang.\*; public class Cds

{

public static void main(String [] args)

{

Clientclass c=new Clientclass(); c.dotask();

}

}

### Server1.java

import java.io.\*; public class Server1

{

public Server1()

{

}

public void runservice()

{

System.out.println("\n Service provided!!!");

}

}

### Clientclass.java:

import java.io.\*; public class Clientclass

{

private Printserver print;

private Dispatcher dis=new Dispatcher(); public Printserver thePrintserver;

public Clientclass()

{

}

public void dotask()

{

dis.request\_channel(); dis.establish\_channel()}

}

}

Dispatcher.java

import java.io.\*;

public class Dispatcher

{

private Clientclass client;

private Printserver print=new Printserver(); public Dispatcher()

{

}

public void establish\_channel()

{

System.out.println("\n\n Established!!!"); print.runservice();

}

public void request\_channel()

{

System.out.println("\n\n Registerd!!!!!");

}

}

### Printserver.java

import java.io.\*; public class Printserver

{

private Server1 server1=new Server1(); private Clientclass client;

private Dispatcher dis; public Printserver()

{

}

public void runservice()

{

server1.runservice();

}

}

### Output :

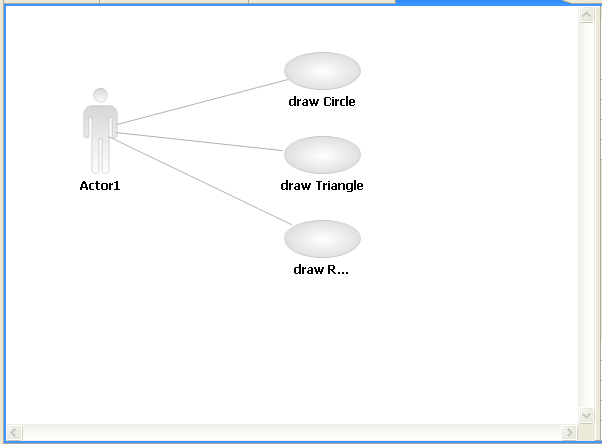
C:\Java>java Cds Registered!!!!!

Established!!!

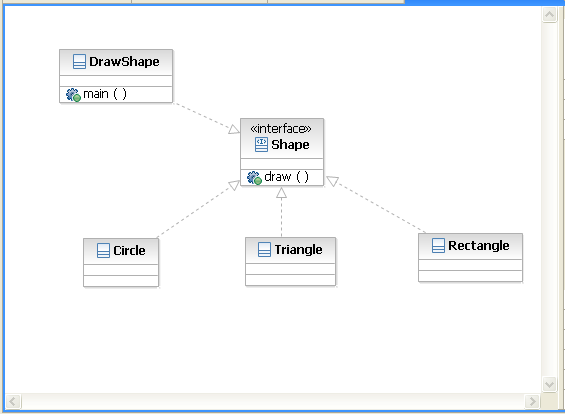
Service provided!!!

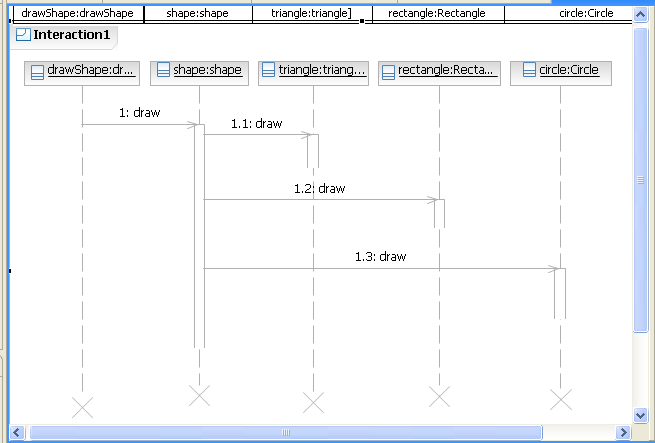
**------------------------------------------------------------------**

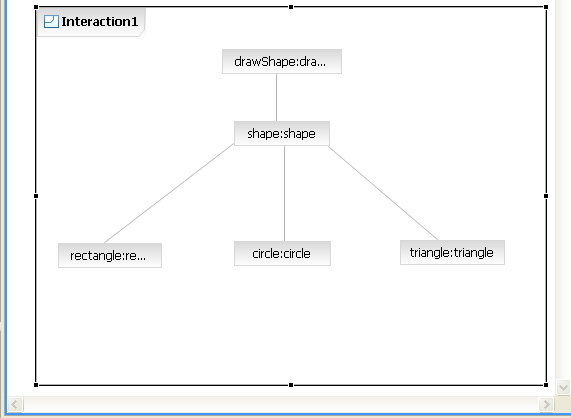
**4.Polymorphism**

Use-Case Diagram

Class Diagram



Sequence Diagram

Collaboration Diagram

#### Implementation

Circle.java

public class Circle implements Shape

{

public void draw()

{

System.*out*.println("This is a circle");

}

}

Rectangle.java

public class Rectangle implements Shape

{

public void draw()

{

System.out*.*println("This is a Rectangle");

}

}

Triangle.java

public class Triangle implements Shape

{

public void draw()

{

System.*out*.println("this is a triangle");

}

}

Shape.java

public interface Shape

{

public void draw();

}

DrawShape.java

import java.util.Scanner; public class DrawShape

{

public static void main(String args[])

{

System.*out*.println("Please enter option draw 1. circle 2. triangle 3.rectangle"); Scanner sin=new Scanner(System.*in*);

int opt;

Shape shape=null; opt=sin.nextInt(); switch(opt)

{

case 1: shape=new Circle();

break;

case 2: shape= new Triangle();

break;

case 3: shape=new Rectangle();

break;

default: System.*out*.println("Invalid option"); System.*exit*(0);

}

shape.draw();

}

}

#### OUTPUT:

Please enter option to draw

1. circle 2. triangle 3.rectangle 1

This is a circle

Please enter option to draw

1. circle 2. triangle 3.rectangle

2

This is a triangle

Please enter option to draw

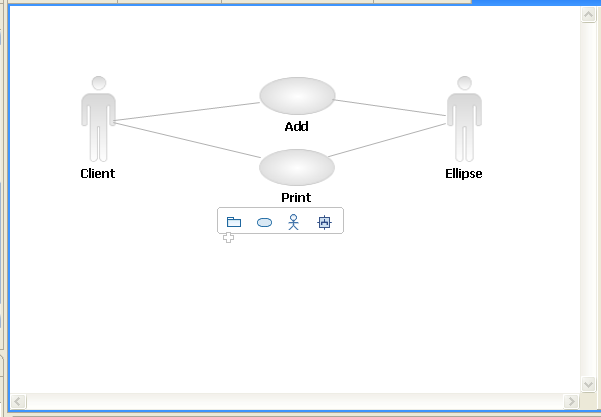
1. circle 2. triangle 3.rectangle

3

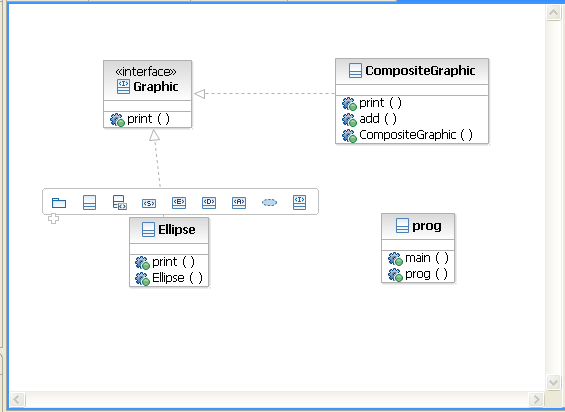
This is a Rectangle

**5.Whole-Part Pattern**

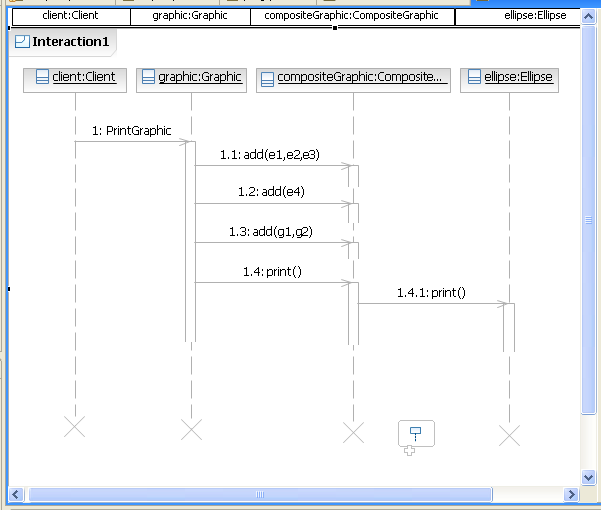
Use-Case Diagram



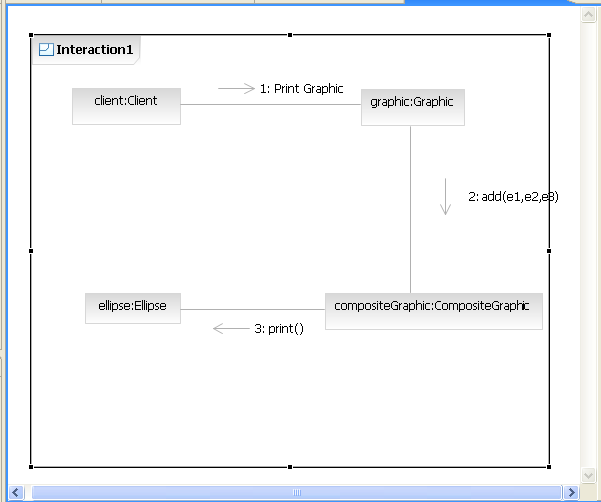
Class Diagram



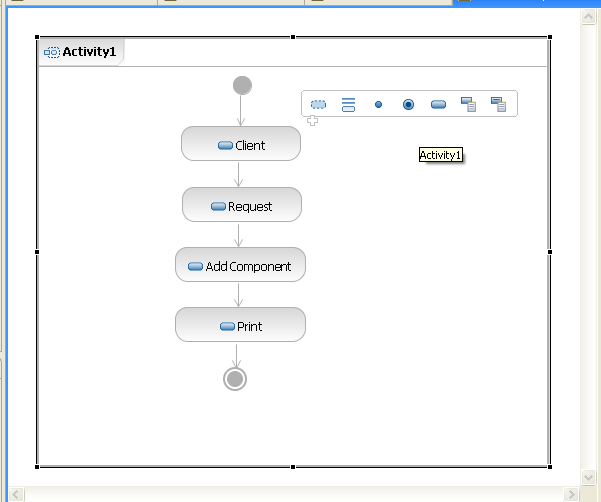
Sequence Diagram



Collaboration Diagram



Activity Diagram



#### Implementation

CompositeGraphic.java

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

public class CompositeGraphic implements Graphic

{

private List <Graphic> childgraphic = new ArrayList<Graphic>(); public void print()

{

for(Graphic graphic : childgraphic)

{

graphic.print();

}

}

public void add(Graphic graphic)

{

childgraphic.add(graphic);

}

}

Ellipse.java

public class Ellipse implements Graphic

{

public void print()

{

System.*out*.println("Ellipse");

}

}

Graphic.java

import java.util.List; import java.util.ArrayList; public interface Graphic

{

public void print();

}

Prog.java

public class prog

{

public static void main(String args[])

{

Ellipse e1 = new Ellipse(); Ellipse e2 = new Ellipse(); Ellipse e3 = new Ellipse(); Ellipse e4 = new Ellipse();

CompositeGraphic g = new CompositeGraphic(); CompositeGraphic g1 = new CompositeGraphic(); CompositeGraphic g2 = new CompositeGraphic(); g1.add(e1);

g1.add(e2);

g1.add(e3);

g2.add(e4);

g.add(g1);

g.add(g2);

g.print()

}

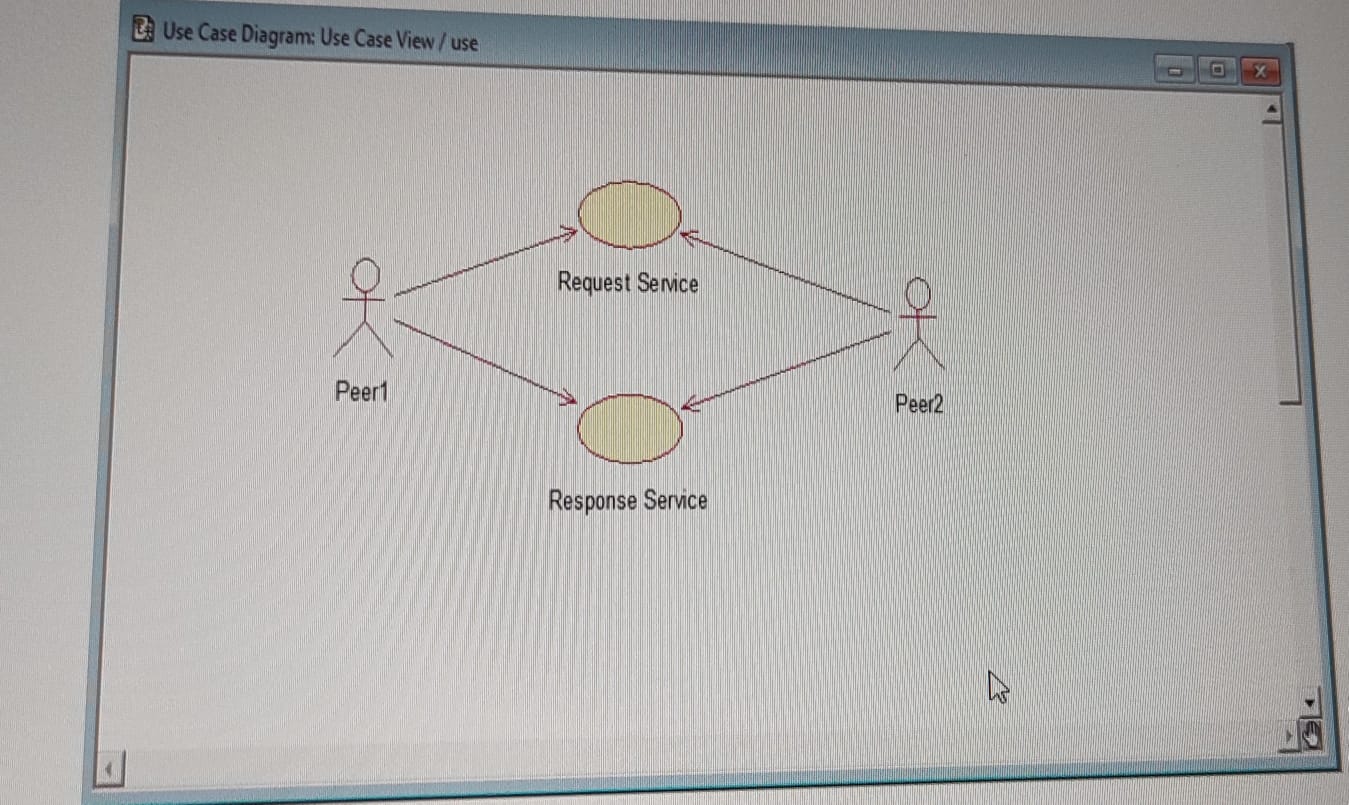
}

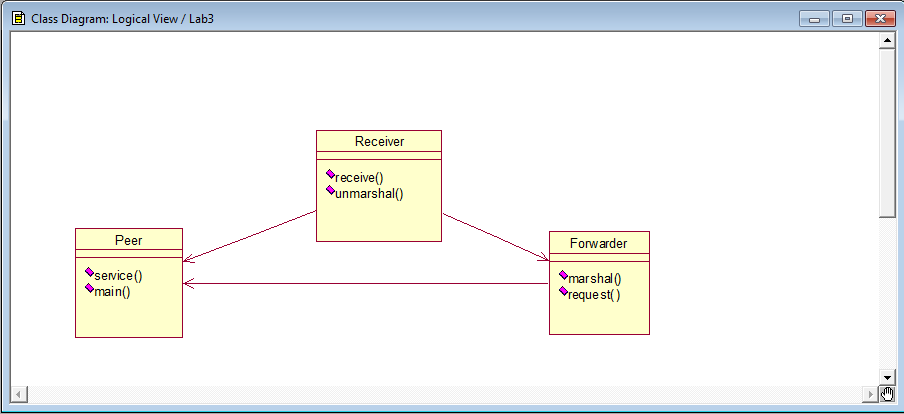
#### OUTPUT

Ellipse Ellipse Ellipse

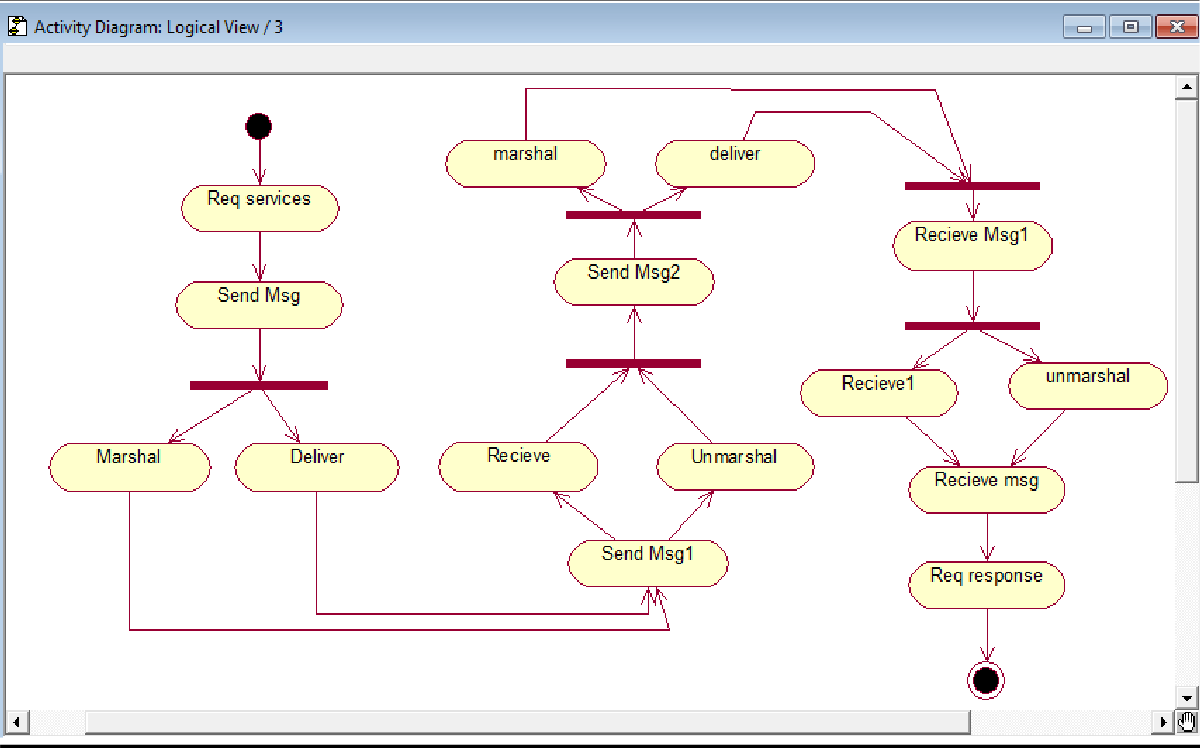
**6.** **Forwarder –Receiver Pattern**

**Use case**



Class diagram

Activity diagram



## Implementation:

### Reciever.java

public class Reciever

{

public void recieve()

{

System.out.println("message received");

}

public void unmarshal()

{

System.out.println("unmarshal done"); recieve();

}

}

Forwarder.java

import java.lang.\*; import java.util.\*; public class Forwarder

{

public Reciever r=new Reciever(); public void marshal()

{

System.out.println("marshal done");

r.unmarshal();

}

}

### Peer.java

public class Peer

{

public Reciever theReciever;

public Forwarder forword=new Forwarder(); public void service()

{

System.out.println("sending message"); forword.marshal();

}

public static void main(String args[])

{

new Peer().service();

}

}

### Output:

C:\Java>java Peer sending message marshal done unmarshal done message received