5. Forwarder Receiver Pattern (Communication)

Intent:
The Forwarder-Receiver design pattern provides transparent inter process communication for software systems with a peer-to-peer interaction model. It introduces forwarders and receivers to decouple peers from the underlying communication mechanisms.

Structure:

![Diagram of Forwarder Receiver Pattern]

Participant Classes:

**Peer** components are responsible for application tasks. To carry out their tasks peers need to communicate with other peers.

**Forwarder** components are responsible for forwarding all these messages to remote network agents without introducing any dependencies on the underlying IPC mechanisms.

**Receiver** components are responsible for receiving messages. A receiver offers a general interface that is an abstraction of a particular IPC mechanism. It includes functionality for receiving and unmarshaling messages.

**Example**: A simple peer-to-peer message exchange scenario; Underlying communication protocol is TCP/IP.
Use Case Diagram:

Sequence Diagram:

Collaboration Diagram:
Activity Diagram:

Class Diagram:
Java Code:

Forwarder.java:

```java
default package FAR;
import java.io.*;
import java.net.Socket;
public class Forwarder {
    private Socket s;
    private OutputStream oStr;

    public byte[] marshal(String theMsg){
        return theMsg.getBytes();
    }

    public void deliver(String theDest, int port, byte[] data) {
        try {
            s = new Socket(theDest, port);
            oStr = s.getOutputStream();
            oStr.write(data);
            oStr.flush();
            oStr.close();
            s.close();
        } catch (IOException e) {
            System.out.println("For error------"+e);
        }
    }
```

Receiver.java

package FAR;
import java.io.IOException;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

public class Receiver {
    private ServerSocket srvS;
    private Socket s;
    private InputStream iStr;
    public Receiver(int port) {
        try {
            srvS = new ServerSocket(port);
            System.out.println("Server started");
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private String unmarshal (byte [] anarray) {
        String message = null;
        message = new String(anarray);
        System.out.println("message " + message.trim());
        return message;
    }

    private byte[] receive() {
        int val;
        byte buffer [] = null;
        try {
            s = srvS.accept();
            iStr = s.getInputStream();
            val = iStr.read();
            buffer = new byte[val];
            iStr.read(buffer);
        } catch (IOException e) {
            System.out.println("Error" + e);
        }
        return buffer;
    }

    public void closeAllSocket() throws IOException {
        iStr.close();
    }
}
s.close();
srvS.close();
}
public String receiveMsg() throws IOException{
    if(unmarshal(receive()).equalsIgnoreCase("close")){
        closeAllSocket();
    }
    return unmarshal(receive());
}

ReceiveThread.java:

package FAR;
public class ReceiveThread implements Runnable {
    String result="";
    public int port=0;
    public String peer="",name="";
    Thread t;
    public ReceiveThread(String threadname,int port) {
        name=threadname;
        t=new Thread(this,name);
        System.out.println("New Thread: "+t);
        this.port=port;
        this.peer=name;
        System.out.println("port and peer "+port+" "+peer);
        t.start();
    }
    @Override
    public void run() {
    try{
        Receiver r=new Receiver(port);
        while(!result.equalsIgnoreCase("close")){
            result =r. receiveMsg();
            System.out.println("Message Received is "+result.trim());
        }
        r.closeAllSocket();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    }
    }

Peer1.java

package FAR;
import java.io.IOException;
import java.util.Scanner;
public class Peer1 {
    Forwarder f=new Forwarder();
    String result="";
    public void execute() throws InterruptedException, IOException{
        Scanner s = new Scanner(System.in);
        String inp="";}
while(true){
    inp=s.next();
    f.sendMsg("127.0.0.1",5200, " "+inp);
}

public static void main(String args[]) throws InterruptedException, IOException{
    new ReceiveThread("Peer1",5500);
    new Peer1().execute();
}

Peer2.java

package FAR;
import java.io.IOException;
import java.util.Scanner;
public class Peer2 {
    Forwarder f=new Forwarder();
    String result="";
    public void execute() throws InterruptedException, IOException{
        Scanner s = new Scanner(System.in);
        String inp="";
        while(true){
            inp=s.next();
            f.sendMsg("127.0.0.1",5500," "+inp);
        }
    }
    public static void main(String args[]) throws InterruptedException, IOException{
        new ReceiveThread("Peer2",5200);
        new Peer2().execute();
    }
}

Output:
Run Peer1.java
Run Peer2.java