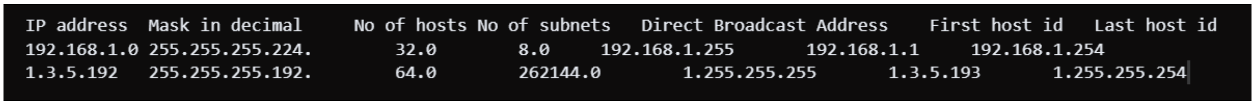
Forwarding Table:

```
public class ip{
 public static char class_type(String address){
 int i=0;
 while(address.charAt(i)!='.'){
 i++;
 }
 int n=Integer.parseInt(address.substring(0,i));
 if(n<128)
 return 'A';
 if(n<192)
 return 'B';
 if(n<224)
 return 'C';
 return 'D';
 }
 public static double[] no_of_hosts(String mask,char cla){
 int c=0;
 for(int i=0;i<mask.length();i++){
 if(mask.charAt(i)=='1')
 c++;
 }
 int hosts=32-c;
 int subnets=0;
 if(cla=='A'){
 subnets=c-8;
 }
 else if(cla=='B'){
 subnets=c-16;
 }
 else if(cla=='C'){
 subnets=c-24;
 }
 return new double[]{Math.pow(2,hosts),Math.pow(2,subnets)};
 }
 public static String decimal_mask(String mask){
 String[] m=mask.split("\\.");
 String d="";
 for(int i=0;i<4;i++){
 int a=Integer.parseInt(m[i],2);
 d=d+String.valueOf(a)+".";
 }
 return d;
 }
 public static String[] addresses(String ip,char cla){
 String[] a=ip.split("\\.");
 String dba="";
 String first="";
 String last="";
 if(cla=='A'){
 dba=a[0]+".255.255.255";
 last=a[0]+".255.255.254";
 }
 else if(cla=='B'){
 dba=a[0]+"."+a[1]+".255.255";
 last=a[0]+"."+a[1]+".255.254";
 }
 else if(cla=='C'){
 dba=a[0]+"."+a[1]+"."+a[2]+".255";
 last=a[0]+"."+a[1]+"."+a[2]+".254";
 }
 int x=Integer.parseInt(a[3]);
 first=a[0]+"."+a[1]+"."+a[2]+"."+String.valueOf(x+1);
 return new String[]{dba,last,first};
}
public static void main(String[] args) {
 String[] ip =new String[]{"192.168.1.0","1.3.5.192"};
 String [] mask=new String[]
{"11111111.11111111.11111111.11100000","11111111.11111111.11111111.11000000"};
 System.out.println("IP address"+"\t"+"Mask in decimal"+"\t"+"\tNo of hosts"+"\t"+"No of
subnets"+"\t"+"Direct Broadcast Address"+"\t"+"First host id"+"\t"+"Last host id");
 for(int i=0;i<ip.length;i++){
 char cla=class_type(ip[i]);
 double [] a=no_of_hosts(mask[i],cla);
 String [] add=addresses(ip[i],cla);

System.out.println(ip[i]+"\t"+decimal_mask(mask[i])+"\t\t"+a[0]+"\t\t"+a[1]+"\t\t"+add[0]+"\t\t"+add[
2]+"\t\t"+add[1]);
 }
 }
}
```



```
IP address  Mask in decimal     No of hosts No of subnets   Direct Broadcast Address    First host id    Last host id
192.168.1.0 255.255.255.224.        32.0        8.0     192.168.1.255       192.168.1.1     192.168.1.254
1.3.5.192   255.255.255.192.        64.0        262144.0        1.255.255.255       1.3.5.193       1.255.255.254
```

Hamming Code:

Client:

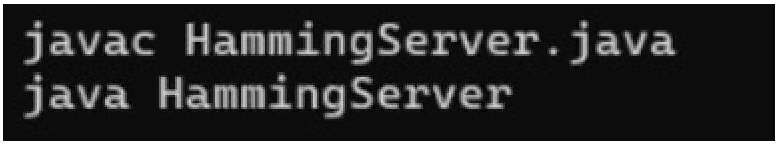
```
import java.io.*;
```

```java
import java.net.*;

public class HammingClient {
    public static void main(String[] args) {
        String serverAddress = "localhost";
        int serverPort = 12345;

        try {
            // Create a socket and connect to server
            Socket socket = new Socket(serverAddress, serverPort);

            // Create input and output streams
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            // Generate data and Hamming code
            String data = "110101";
            String hammingCode = generateHammingCode(data);
            System.out.println("Sending Hamming code: " + hammingCode);

            // Send Hamming code to server
            out.println(hammingCode);

            // Receive response from server
            String response = in.readLine();
            System.out.println("Server response: " + response);

            // Close streams and socket
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Hamming code generation function
    private static String generateHammingCode(String data) {
        int n = data.length();
        int m = 0;

        // Find the number of parity bits
        while ((1 << m) < (n + m + 1)) {
            m++;
        }

        StringBuilder sb = new StringBuilder(n + m);
        int j = 0;

        for (int i = 0; i < n + m; i++) {
            if (((i + 1) & i) == 0) {
                sb.append('0');
            } else {
                sb.append(data.charAt(j++));
            }
        }

        for (int i = 0; i < m; i++) {
            int parityIndex = (1 << i) - 1;
            int count = 0;
            for (int k = parityIndex; k < n + m; k += (1 << (i + 1))) {
                for (int l = k; l < Math.min(k + (1 << i), n + m); l++) {
                    if (sb.charAt(l) == '1') {
                        count++;
                    }
                }
            }
            sb.setCharAt(parityIndex, (count % 2 == 0) ? '0' : '1');
        }

        return sb.toString();
    }
}
```

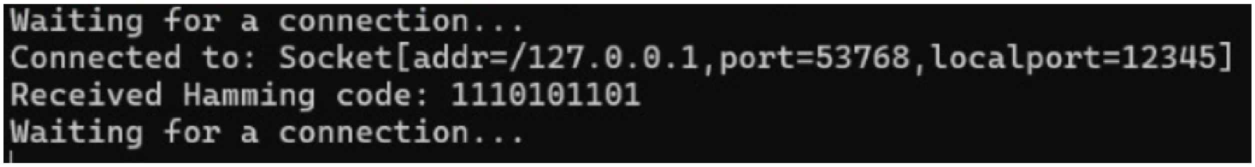
Server:

```
import java.io.*;
import java.net.*;

public class HammingServer {
    public static void main(String[] args) {
        int port = 12345;

        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(port);

            while (true) {
                System.out.println("Waiting for a connection...");
                // Accept client connection
                Socket clientSocket = serverSocket.accept();
                System.out.println("Connected to: " + clientSocket);

                // Create input and output streams
                BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                // Receive Hamming code from client
                String hammingCode = in.readLine();
                System.out.println("Received Hamming code: " + hammingCode);

                // Perform error checking on Hamming code
                boolean hasError = checkHammingCode(hammingCode);

                // Send response to client
                if (hasError) {
                    out.println("Bad data");
                } else {
                    out.println("Good data");
                }

                // Close streams and socket
                in.close();
                out.close();
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Hamming code error checking function
    private static boolean checkHammingCode(String hammingCode) {
        int n = hammingCode.length();
        int m = 0;

        // Find the number of parity bits
        while ((1 << m) < (n + m + 1)) {
            m++;
        }
        int[] parityPositions = new int[m];
        for (int i = 0; i < m; i++) {
            parityPositions[i] = (1 << i) - 1;
        }

        boolean hasError = false;

        for (int i = 0; i < m; i++) {
            int parityIndex = (1 << i) - 1;
            int count = 0;
            for (int j = parityIndex; j < n; j += (1 << (i + 1))) {
                for (int k = j; k < Math.min(j + (1 << i), n); k++) {
                    if (hammingCode.charAt(k) == '1') {
                        count++;
                    }
                }
            }
            if (count % 2 != 0) {
                hasError = true;
                break;
            }
        }

        return hasError;
    }
}
```

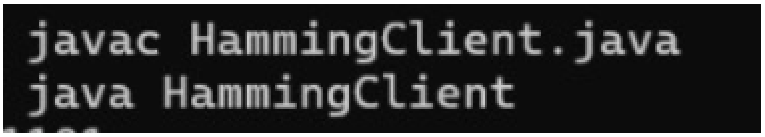
Output:



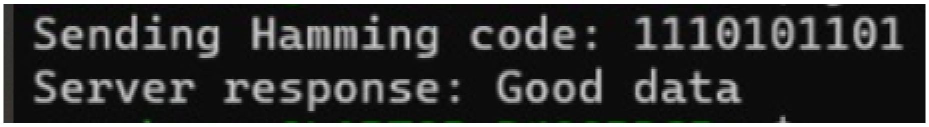
```
javac HammingServer.java
java HammingServer
```



```
Waiting for a connection...
Connected to: Socket[addr=/127.0.0.1,port=53768,localport=12345]
Received Hamming code: 1110101101
Waiting for a connection...
```



```
javac HammingClient.java
java HammingClient
```



```
Sending Hamming code: 1110101101
Server response: Good data
```