```java
/* To implement Fiat-Shamir protocol for entity authentication
   using a client server program where the client is the
   claimant and the server is the verifier.
*/

// CLAIMANT SIDE

import java.net.*;
import java.io.*;
import java.util.*;
public class Claimant
{
    // initialise socket and input output streams
    private Socket socket        = null;
    private DataInputStream input = null;
    private DataInputStream inFromVerifier   = null;
    private DataOutputStream out     = null;

    //public key of verifier, private key of claimant, public key of claimant,
      witness, commitment, challenge from verifier, response to the challenge
    static int n, s, v, x, r, c, y;
    Scanner scan = new Scanner(System.in);

    static int gcd(int a, int b) {
        int t;
        while(b != 0){
            t = a; a = b; b = t%b;
        }
        return a;
    }
    static boolean relativelyPrime(int a, int b) {
        return gcd(a,b) == 1;
    }
    public Claimant(String address, int port)
    {
        try
        {
            socket = new Socket(address, port);
            System.out.println("\nConnected with the verifier.");
    // takes input from terminal
    input = new DataInputStream(System.in);

    // sends output to the socket
    out = new DataOutputStream(socket.getOutputStream());

    // takes input from the server socket
    inFromVerifier = new DataInputStream(
    new BufferedInputStream(socket.getInputStream()));
}
catch(UnknownHostException u)
{
    System.out.println(u);
}
catch(IOException i)
{
    System.out.println(i);
}

String line = "";

while (!line.equals("Over"))
{
    try
    {
        // reading public key of verifier
            line = inFromVerifier.readUTF();
            // parsing integer from string
            n = Integer.parseInt(line);
        System.out.print("\nVerifier:: Public key -> "+line);

        // choosing a private key
        int nMinusOne = n - 1;
        System.out.print("\nChoose a number as private key.\n(");
        for(int i = 2; i < n; i++)
        {
            if (relativelyPrime(i, n))
            {
                System.out.print(i+" ");
            }
```

```
                }
                System.out.print("\b)\n-> ");
                s = scan.nextInt();

                //calculating public key
                int temp = 1;
                for (int i = 1; i <=2; i++)
                {
                    temp = temp * s; // calculating square of s
                }
                v = temp % n;

                // registering public key with verifier
                System.out.print("Claimant:: ");
                out.writeUTF(Integer.toString(v));
                System.out.print(v+" (registering public key)");

                // choose commitment 'r'
                System.out.print("\nChoose a number as commitment (between 1 and
"+nMinusOne+"):: " );
                r = scan.nextInt();

                // calculating witness
                temp = 1;
                for (int i = 1; i <=2; i++)
                {
                    temp = temp * r; // calculating square of r
                }
                x = temp % n;

                // sending x to the verifier
                System.out.print("Claimant:: ");
                out.writeUTF(Integer.toString(x));
                System.out.print(x+" (sending commitment)");

                // input of challenge from verifier
                System.out.print("\nVerifier:: ");
                line = inFromVerifier.readUTF();
                c = Integer.parseInt(line);
                System.out.print(Integer.toString(c)+" (challenge)");
```

```java
                // sending response to verifier 'y = r*s^c'
                System.out.print("\nClaimant:: ");
                temp = 1;
                for (int i = 1; i <=c; i++)
                { temp = temp * s; // calculating exponent of s }

                y = (r * temp);
                out.writeUTF(Integer.toString(y));
                System.out.print(y+" (sending response)");

                // waiting for verification
                System.out.print("\nVerifier:: ");
                line = inFromVerifier.readUTF();
                System.out.print(line);
                System.out.print("\n\n");
                break;
            }
            catch(IOException i)
            { System.out.println(i); }
        }
        try
        {
            input.close();
            inFromVerifier.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Claimant claimant = new Claimant("127.0.0.1", 8221);
    }
}
```