```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package identity_server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Security;
import java.util.Random;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;


    // Lit un entier depuis l'entrée standard

/**
 *
 * @author rapha
 */
public class IdentityApplic {
    public static void main(String[] args)
            throws UnknownHostException, IOException, ClassNotFoundException,
            NoSuchAlgorithmException, NoSuchPaddingException,
            IllegalBlockSizeException, BadPaddingException, InvalidKeyException
    {
        Security.addProvider(
            new org.bouncycastle.jce.provider.BouncyCastleProvider()
        );

        Socket sock = new Socket(Config.IDENTITY_SERVER, Config.IDENTITY_PORT);
        ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(sock.getInputStream());

        SecretKey sessionKey = keyExchange(in, out);

        // Crée les instances de cryptage et de décryptage
        Cipher cryptor = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cryptor.init(Cipher.ENCRYPT_MODE, sessionKey);
        Cipher decryptor = Cipher.getInstance("DES/ECB/PKCS5Padding");
        decryptor.init(Cipher.DECRYPT_MODE, sessionKey);

        login(in, out, cryptor, decryptor);
    }

    public static int readInt() throws IOException
    {
        BufferedReader inStream = new BufferedReader (
            new InputStreamReader(System.in)
        );
        return Integer.parseInt(inStream.readLine());
    }

    // Lit une lige depuis l'entrée standard
    public static String readLine() throws IOException
    {
        BufferedReader inStream = new BufferedReader (
            new InputStreamReader(System.in)
        );
        return inStream.readLine();
    }

    public static SecretKey keyExchange(ObjectInputStream in,
            ObjectOutputStream out)
            throws IllegalBlockSizeException, BadPaddingException,
            InvalidKeyException, NoSuchAlgorithmException, IOException,
            ClassNotFoundException, NoSuchPaddingException
    {
        // Génère une paire de clé pour la réception de la clé de cryptage
        // asynchrone
        KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA");
        gen.initialize(1024, new SecureRandom());
        KeyPair keys = gen.generateKeyPair();
        PublicKey publicKey = keys.getPublic();
        PrivateKey privateKey = keys.getPrivate();

        // Envoie la clé publique au serveur
        out.writeObject(new KeyExchangeClient(publicKey));
        out.flush();

        // Recoit la clé de session cryptée par le serveur
        KeyExchangeServer response = (KeyExchangeServer) in.readObject();

        // Décrypte la clé de session avec la clé privée
        //Cipher decryptor = Cipher.getInstance("RSA/ECB/PKCS#1");
        Cipher decryptor = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        decryptor.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] sessionKeyEncoded =
                decryptor.doFinal(response.getCryptedSessionKey());
        return new SecretKeySpec(sessionKeyEncoded, "DES");

    }

    private static void login(ObjectInputStream in, ObjectOutputStream out,
            Cipher cryptor, Cipher decryptor)
            throws IOException, ClassNotFoundException,
            IllegalBlockSizeException, BadPaddingException,
            NoSuchAlgorithmException
    {
        // Reçoit et décrypte le sel de hashage du serveur
        LoginServer saltQuery = (LoginServer) Utils.decryptObject(
                (byte[]) in.readObject(), decryptor
        );
        System.out.println("Salt: "+ saltQuery.getHashSalt());

        System.out.println("Nom d'utilisateur: ");
        String user = readLine();
        System.out.println("Mot de passe: ");
        String pass = readLine();

        // Envoie les informations d'authentification avec le
        // mot de passe hashé
        int clientSalt = (new Random()).nextInt();
        out.writeObject(
            Utils.cryptObject(
                new LoginClient(
                    user, Utils.hashPassword(
                        pass, clientSalt, saltQuery.getHashSalt()
                    ), clientSalt
                ), cryptor
            )
        );
        out.flush();

        // Reçoit la réponse
        Protocol loginResponse = (Protocol) Utils.decryptObject(
            (byte[]) in.readObject(), decryptor
        );
        if (loginResponse instanceof Ack) {
            System.out.println("Connexion réussie");
            verifId(in, out, cryptor, decryptor);
        } else if (loginResponse instanceof Fail) {
            System.out.println("Mauvais identifiants");
        }
    }

    private static void verifId(ObjectInputStream in, ObjectOutputStream out,
            Cipher cryptor, Cipher decryptor)
            throws IOException, IllegalBlockSizeException, BadPaddingException,
            ClassNotFoundException
    {
        for (;;) {
            System.out.println("Nom du client: ");
            String clientName = readLine();
            System.out.println("Prénom du client: ");
            String clientSurname = readLine();
            System.out.println("Numéro national du client: ");
            int clientNationalID = readInt();

            out.writeObject(
                Utils.cryptObject(
                    new VerifId(clientName, clientSurname, clientNationalID),
                    cryptor
                )
            );
            out.flush();

            Protocol response = (Protocol) Utils.decryptObject(
                (byte[]) in.readObject(), decryptor
            );

            if (response instanceof Ack) {
                System.out.println("Identité valide");
            } else if (response instanceof Fail) {
                System.out.println("Identité non valide");
            }
        }
    }
}
```