ITAM

Sistemas Distribuidos

Mariana Hernández Rocha - 150845
Maestría en Ciencias en Computación

Fabián Orduña Ferreira - 159001
Maestría en Ciencias en Computación

Salvador García González - 119718
Maestría en Ciencias en Computación

Primer Proyecto: Segunda entrega

Encrypted Remote Method Invocation

Primavera 2021

Profesor:
Marcelo Mejía Olvera

Capítulo 1

Aplicación RMI

En este proyecto se implementa una aplicación usando RMI que ejecuta solicitudes SQL de varios clientes a un servidor que tiene montado una base de datos MySQL. La aplicación fue creada usando Java 8 en complemento con las siguientes bibliotecas:

- commons-dbcp
- commons-logging
- commons-pool
- mysql-connector-java

Como IDE se utiliza Netbeans 8, que nos permite contar con un proyecto que contenga todos los códigos.

El caso de uso que se implementa es un sistema de aerolínea que da respuesta a distintas usos comúnes. Se emplea una base de datos en MySQL que contiene 4 tablas:

- **lugar**: contiene el id_lugar y el nombre de los lugares que pueden ser origen o destino
- **persona**: contiene el id_persona y el nombre de las personas que han tomado o tomarán un vuelo
- **persona_vuelo**: contiene la relación de id_persona y id_vuelos
- **vuelo**: contiene el id_vuelo, así como su id_origen, id_destino y la fecha del vuelo

La idea es que los clientes puedan consultar al servidor que tiene levantada la base de datos y esta responda de acuerdo al método invocado.

1.0.1. Pool de conexiones - ConnectionPool.java

```java
package bd;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import org.apache.commons.dbcp2.BasicDataSource;
/**
 *
 * @author mcc06
 */
public class ConnectionPool {
    private final String DB="computo_distribuido";
    private final String URL="jdbc:mysql://localhost:3306/"+DB+"?useUnicode=true"
            + "&useJDBCCompliantTimezoneShift=true&"
            + "useLegacyDatetimeCode=false&serverTimezone=UTC";
    private final String USER="root";
    private final String PASS="";

    private static ConnectionPool dataSource;
    private BasicDataSource basicDataSource=null; //es el que pertmite crearlo

    private ConnectionPool(){

        basicDataSource = new BasicDataSource();
        basicDataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        basicDataSource.setUsername(USER);
        basicDataSource.setPassword(PASS);
        basicDataSource.setUrl(URL);

        basicDataSource.setMinIdle(5);
        basicDataSource.setMaxIdle(20);
        basicDataSource.setMaxTotal(50);
        basicDataSource.setMaxWaitMillis(-1);

    }

    public static ConnectionPool getInstance() {
        if (dataSource == null) {
            dataSource = new ConnectionPool();
            return dataSource;
        } else {
            return dataSource;
        }
    }

    public Connection getConnection() throws SQLException{
      return this.basicDataSource.getConnection();
    }

    public void closeConnection(Connection connection) throws SQLException {
        connection.close();
    }

    PreparedStatement prepareStatement(String sql) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

1.1. RMI

RMI implementa dos clases: Client.java, Server.java y adicionalmente implementa una interfaz: Hello.java. Además se especifíca otra clase ModeloVuelos.java la cual contiene los métodos que se pueden invocar.

1.1.1. Interfaz - ModeloVuelos.java

En el archivo ModeloVuelos.java se importan las librerias, se declara la conexión y se declaran los métodos que el cliente podrá ejecutar:

1 - Obtener todos los destinos a los cuales viaja la aerolinea

```
1     public ArrayList<Lugar> obtenerLugares() throws SQLException{
2         ArrayList<Lugar> resultados = new ArrayList();
3
4         String sql = "SELECT DISTINCT id_lugar, nombre FROM lugar";
5         Lugar l;
6         ResultSet resultado;
7         try{
8             PreparedStatement statement = c.prepareStatement(sql);
9             resultado = statement.executeQuery();
10            System.out.println(resultado.toString());
11
12            while(resultado.next()){
13                int resIdLugar = Integer.parseInt(resultado.getString("id_lugar"));
14                String resNombre = resultado.getString("nombre");
15                l = new Lugar(resIdLugar, resNombre);
16                resultados.add(l);
17            }
18        }catch(SQLException e){
19            System.out.println(e.toString());
20            return resultados;
21        }
22        return resultados;
23    }
```

2 - Obtener las personas que abordarán un vuelo

```
1  public ArrayList<Persona> obtenerPersonasVuelo(int idVuelo){
2          ArrayList<Persona> resultados = new ArrayList();
3          String sql = "SELECT * FROM persona WHERE persona.id_persona in "
4                  + "(SELECT persona_vuelo.id_persona FROM persona_vuelo"+
5                  " WHERE persona_vuelo.id_vuelo = ?)";
6          ResultSet resultado;
7          Persona p;
8          try{
9              PreparedStatement statement = c.prepareStatement(sql);
10             statement.setInt(1,idVuelo);
11             resultado = statement.executeQuery();
12             System.out.println(resultado.toString());
13             while(resultado.next()){
14                 int resIdPersona = Integer.parseInt(resultado.getString("id_persona"));
15                 String resNombrePersona = resultado.getString("nombre");
16                 p= new Persona(resIdPersona, resNombrePersona);
17                 resultados.add(p);
18             }
19         }
20         catch(Exception e){
21             System.out.println(e.toString());
22             return resultados;
23         }
24         return resultados;
25     }
```

3 - Obtener todos los datos registrados de un vuelo

```
public Vuelo obtenVuelo(int idVuelo) throws SQLException{

        String sql = "SELECT * FROM vuelo WHERE id_vuelo = ?";
        ResultSet resultado;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setInt(1,idVuelo);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            resultado.next();
            //(int idVuelo, int idOrigen, int idDestino, String fecha)
            int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
            int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
            int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
            String resFecha = resultado.getString("fecha");
            return new Vuelo(resIdVuelo, resIdOrigen, resIdDestino, resFecha);
        }catch(SQLException e){
            System.out.println(e.toString());
            return null;
        }
    }
```

4 - Obtener todos los vuelos registrados de un pasajero anteriores a una fecha

```
public ArrayList<Vuelo> vuelosAnteriores Persona(String fecha, int idPersona){
        ArrayList<Vuelo> resultados = new ArrayList();
        String sql =
        "SELECT DISTINCT persona.id_persona, persona.nombre, vuelo.id_vuelo, vuelo.fecha,"
        + "(SELECT lugar.nombre FROM lugar WHERE lugar.id_lugar = vuelo.id_origen) as origen,"
        + "(SELECT lugar.nombre FROM lugar WHERE lugar.id_lugar = vuelo.id_destino) as destino"
        + "FROM persona INNER JOIN persona_vuelo ON persona_vuelo.id_persona = persona.id_persona"
        + "INNER JOIN vuelo ON vuelo.id_vuelo = persona_vuelo.id_vuelo WHERE persona.id_persona = ?"
        + "AND vuelo.fecha <= ?";
        ResultSet resultado;
        Vuelo v;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setInt(1,idPersona);
            statement.setString(2,fecha);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }
        catch(Exception e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

5 - Obtener los vuelos disponibles a partir de una fecha

```java
public ArrayList<Vuelo> vuelosDisponibles(String fecha) throws SQLException{

        ArrayList<Vuelo> resultados = new ArrayList();

        String sql = "SELECT * FROM vuelo WHERE fecha >= ?";
        ResultSet resultado;
        Vuelo v;

        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setString(1,fecha);

            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }catch(SQLException e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

6 - Obtener los vuelos disponibles a partir de una fecha para un pasajero

```java
public ArrayList<Vuelo> vuelosDisponiblesPersona(String fecha, int idPersona){
        ArrayList<Vuelo> resultados = new ArrayList();
        String sql = "SELECT * FROM vuelo WHERE fecha >= ? and id_vuelo in"
        + "( SELECT id_vuelo FROM persona_vuelo WHERE id_persona = ?)";
        ResultSet resultado;
        Vuelo v;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setString(1,fecha);
            statement.setInt(2,idPersona);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }catch(SQLException e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

7 - Obtener todos los vuelos históricos en la aerolínea

```
public ArrayList<Vuelo> vuelosHistoricos(){
        ArrayList<Vuelo> resultados = new ArrayList();
        String sql = "SELECT * FROM vuelo";
        ResultSet resultado;
        Vuelo v;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }
        catch(Exception e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

8 - Obtener todos los vuelos históricos para una persona

```
public ArrayList<Vuelo> vuelosHistoricosPersona(int idPersona) throws SQLException{
        ArrayList<Vuelo> resultados = new ArrayList();
        String sql = "SELECT * FROM vuelo WHERE id_vuelo in"
        + "( SELECT id_vuelo FROM persona_vuelo WHERE id_persona = ?)";
        ResultSet resultado;
        Vuelo v;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setInt(1,idPersona);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }catch(SQLException e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

9 - Obtener todos los vuelos con un origen y un destino específico

```
public ArrayList<Vuelo> vuelosOrigenDestino(int idOrigen, int idDestino){
        ArrayList<Vuelo> resultados = new ArrayList();
        String sql = "SELECT * FROM vuelo WHERE id_origen = ? and id_destino = ?";
        ResultSet resultado;
        Vuelo v;
        try{
            PreparedStatement statement = c.prepareStatement(sql);
            statement.setInt(1,idOrigen);
            statement.setInt(2,idDestino);
            resultado = statement.executeQuery();
            System.out.println(resultado.toString());
            while(resultado.next()){
                int resIdVuelo = Integer.parseInt(resultado.getString("id_vuelo"));
                int resIdOrigen = Integer.parseInt(resultado.getString("id_origen"));
                int resIdDestino = Integer.parseInt(resultado.getString("id_destino"));
                String resFecha = resultado.getString("fecha");
                v= new Vuelo(resIdVuelo, resIdOrigen,resIdDestino, resFecha);
                resultados.add(v);
            }
        }catch(SQLException e){
            System.out.println(e.toString());
            return resultados;
        }
        return resultados;
    }
```

1.1.2. Interfaz - Hello.java

En la interfaz se declaran todos los métodos que va a tener acceso nuestra conexión.

```
1   package rmi;
2
3   import bd.Lugar;
4   import bd.Persona;
5   import bd.Vuelo;
6   import java.rmi.Remote;
7   import java.rmi.RemoteException;
8   import java.sql.SQLException;
9   import java.util.ArrayList;
10
11  /**
12   *
13   * @author mcc06
14   */
15  public interface Hello extends Remote{
16      String sayHello(String persona) throws RemoteException;
17      String sayHello() throws RemoteException;
18      int insertaAlumno(String nombre, String paterno, String materno)
19      throws RemoteException;
20      int actualizaAlumno(int idAlumno, String nombre, String paterno, String materno)
21      throws RemoteException;
22
23      public ArrayList<Vuelo> vuelosHistoricos() throws RemoteException;
24      public ArrayList<Vuelo> vuelosDisponibles(String fecha) throws RemoteException;
25      public Vuelo obtenerVuelo(int idVuelo) throws RemoteException;
26      public ArrayList<Persona> obtenerPersonasVuelo(int idVuelo) throws RemoteException;
27      public ArrayList<Vuelo> vuelosHistoricosPersona(int idPersona) throws RemoteException;
28      public ArrayList<Vuelo> vuelosDisponiblesPersona(String fecha, int idPersona) throws RemoteException;
29      public ArrayList<Vuelo> vuelosAnteriores Persona(String fecha, int idPersona) throws RemoteException;
30      public ArrayList<Lugar> obtenerLugares() throws RemoteException;
31      public ArrayList<Vuelo> vuelosOrigenDestino(int idOrigen, int idDestino) throws RemoteException;
32
33  }
```

Clase Vuelo

```java
package bd;
import java.io.Serializable;

public class Vuelo implements Serializable{
    private int idVuelo;
    private int idOrigen;
    private int idDestino;
    private String fecha;

    public Vuelo(int idVuelo, int idOrigen, int idDestino, String fecha) {
        this.idVuelo = idVuelo;
        this.idOrigen = idOrigen;
        this.idDestino = idDestino;
        this.fecha = fecha;
    }
    public int getIdVuelo() {
        return idVuelo;
    }

    public void setIdVuelo(int idVuelo) {
        this.idVuelo = idVuelo;
    }

    public int getIdOrigen() {
        return idOrigen;
    }

    public void setIdOrigen(int idOrigen) {
        this.idOrigen = idOrigen;
    }

    public int getIdDestino() {
        return idDestino;
    }

    public void setIdDestino(int idDestino) {
        this.idDestino = idDestino;
    }

    public String getFecha() {
        return fecha;
    }

    public void setFecha(String fecha) {
        this.fecha = fecha;
    }

    @Override
    public String toString() {
        return "Vuelo{" + "idVuelo=" + idVuelo + ", idOrigen=" + idOrigen
                + ", idDestino=" + idDestino + ", fecha=" + fecha + '}';
    }
}
```

Clase Persona

```java
package bd;

import java.io.Serializable;

/**
 *
 * @author mcc06
 */
public class Persona implements Serializable{

    private int idPersona;
    private String nombre;

    public Persona(int idPersona, String nombre) {
        this.idPersona = idPersona;
        this.nombre = nombre;
    }

    public int getIdPersona() {
        return idPersona;
    }

    public void setIdPersona(int idPersona) {
        this.idPersona = idPersona;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public String toString() {
        return "Persona{" + "idPersona=" + idPersona + ", nombre=" + nombre + '}';
    }


}
```

Clase Lugar

```java
package bd;

import java.io.Serializable;

/**
 *
 * @author mcc06
 */
public class Lugar implements Serializable{
    private int idLugar;
    private String nombre;

    public Lugar(int idLugar, String nombre) {
        this.idLugar = idLugar;
        this.nombre = nombre;
    }

    public int getIdLugar() {
        return idLugar;
    }

    public void setIdLugar(int idLugar) {
        this.idLugar = idLugar;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public String toString() {
        return "Lugar{" + "idLugar=" + idLugar + ", nombre=" + nombre + '}';
    }
}
```

1.1.3. Cliente - Client.java

Ejecuta el código de cara al cliente:

```
1  package rmi;
2  import bd.Lugar;
3  import bd.Persona;
4  import bd.Vuelo;
5  import java.io.BufferedReader;
6  import java.io.InputStreamReader;
7  import java.rmi.registry.LocateRegistry;
8  import java.rmi.registry.Registry;
9  import java.util.ArrayList;
10
11 public class Client {
12     private Client() {}
13     public static void main(String[] args) {
14         String host = (args.length < 1) ? "148.205.36.206" : args[0];
15         try {
16             System.setProperty("java.rmi.server.hostname", host);
17             //Registry registry = LocateRegistry.getRegistry(1010);
18             Registry registry = LocateRegistry.getRegistry(host, 1010);
19             Hello stub = (Hello) registry.lookup("Hello");
20             System.out.println("Bienvenido al sistema de vuelos de la aerolínea "
21             + " 'Distributed friends'. \n"
22             + "Este sistema te proporciona datos útiles para localizar"
23             + "vuelos, lugares y personas \n"
24             + "registradas en el sistema. El menú principal contiene 9 métodos"
25             + "identificados con números: \n"
26             + "\n"
27             + "1 - Obtener todos los destinos a los cuales viaja la aerolinea \n"
28             + "    No recibe como entrada ningún parámetro  \n"
29             + "2 - Obtener todos los pasajeros registrados en un vuelo \n"
30             + "    Recibe como entrada el id del vuelo a buscar \n"
31             + "3 - Obtener todos los datos registrados de un vuelo \n"
32             + "    Recibe como entrada el id del vuelo a buscar \n"
33             + "4 - Obtener todos los vuelos registrados de un pasajero anteriores"
34             + "a una fecha \n"
35             + "    Recibe como entrada el id del pasajero y la fecha máxima de viaje\n"
36             + "5 - Obtener los vuelos disponibles a partir de una fecha\n"
37             + "    Recibe como entrada la fecha a partir de la cual buscar \n"
38             + "6 - Obtener los vuelos disponibles a partir de una fecha para un"
39             + "pasajero\n"
40             + "    Recibe como entrada el id del pasajero y la fecha a partir"
41             + "de la cual buscar \n"
42             + "7 - Obtener todos los vuelos históricos en la aerolínea\n"
43             + "    No recibe como entrada ningún parámetro \n"
44             + "8 - Obtener todos los vuelos históricos para una persona\n"
45             + "    Recibe como entrada el id del pasajero  \n"
46             + "9 - Obtener todos los vuelos con un origen y un destino específico\n"
47             + "    Recibe como entrada el id del lugar de origen y el id "
48             + " del lugar destino\n"
49             + "\n"
50             + "Escribe el número del método a ejecutar seguido de los parámetros"
51             + "indicados:");
52
53             BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
54             int opcion0, opcion = Integer.parseInt(reader.readLine());
55             String cadenaOpcion = "";
56
57             try{
58                 switch(opcion){
59                     case 1:
60                         ArrayList<Lugar> lugares = stub.obtenerLugares();
61                         System.out.println(lugares.toString());
62                         break;
63                     case 2:
64                         System.out.println("Escribe el número de vuelo");
```

```
65                      opcion = Integer.parseInt(reader.readLine());
66                      ArrayList<Persona> personas = stub.obtenerPersonasVuelo(opcion);
67                      System.out.println(personas.toString());
68                      break;
69                  case 3:
70                      System.out.println("Escribe el número de vuelo");
71                      opcion = Integer.parseInt(reader.readLine());
72                      Vuelo vuelo = stub.obtenerVuelo(opcion);
73                      System.out.println(vuelo.toString());
74                      break;
75                  case 4:
76                      System.out.println("Escribe el id de la persona");
77                      opcion = Integer.parseInt(reader.readLine());
78                      System.out.println("Escribe la fecha");
79                      cadenaOpcion = reader.readLine();
80                      ArrayList<Vuelo> vuelosAntPersona =
81                      stub.vuelosAnteriores Persona(cadenaOpcion, opcion);
82                      System.out.println(vuelosAntPersona.toString());
83                      break;
84                  case 5:
85                      System.out.println("Escribe la fecha (aaaa-mm-dd)");
86                      cadenaOpcion = reader.readLine();
87                      ArrayList<Vuelo> vuelosDisp =
88                      stub.vuelosDisponibles(cadenaOpcion);
89                      System.out.println(vuelosDisp.toString());
90                      break;
91                  case 6:
92                      System.out.println("Escribe el id de la persona");
93                      opcion = Integer.parseInt(reader.readLine());
94                      System.out.println("Escribe la fecha");
95                      cadenaOpcion = reader.readLine();
96                      ArrayList<Vuelo> vuelosPersona =
97                      stub.vuelosDisponiblesPersona(cadenaOpcion, opcion);
98                      System.out.println(vuelosPersona.toString());
99                      break;
100                 case 7:
101                     ArrayList<Vuelo> vuelosHist = stub.vuelosHistoricos();
102                     System.out.println(vuelosHist.toString());
103                     break;
104                 case 8:
105                     System.out.println("Escribe el id de la persona");
106                     opcion = Integer.parseInt(reader.readLine());
107                     ArrayList<Vuelo> vuelosHistPersona =
108                     stub.vuelosHistoricosPersona(opcion);
109                     System.out.println(vuelosHistPersona.toString());
110                     break;
111                 case 9:
112                     System.out.println("Escribe el id del origen");
113                     opcion = Integer.parseInt(reader.readLine());
114                     System.out.println("Escribe el id del destino");
115                     opcion0 = Integer.parseInt(reader.readLine());
116                     ArrayList<Vuelo> vuelosOrigenDestino =
117                     stub.vuelosOrigenDestino(opcion, opcion0);
118                     System.out.println(vuelosOrigenDestino.toString());
119                     break;
120                 default:
121                     System.out.println("Opción no encontrada");
122                     break;
123             }
124         }catch(Exception e){
125             System.out.println("Error");
126             System.out.println(e.toString());
127         }
128         System.out.println("BYE =)");
129     } catch (Exception e) {
130         System.err.println("Client exception: " + e.toString());
131         e.printStackTrace();
132     }
```

```
133     }
134 }
```

1.1.4. Servidor - Server.java

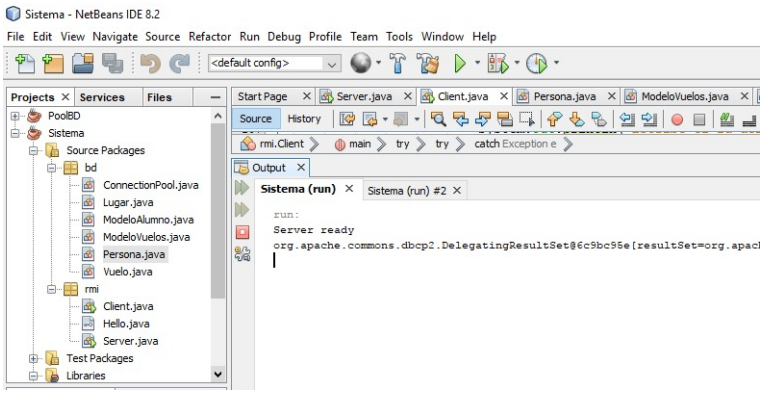
```
1  package rmi;
2  import bd.Lugar;
3  import bd.ModeloAlu mno;
4  import bd.ModeloVuelos;
5  import bd.Persona;
6  import bd.Vuelo;
7  import java.rmi.registry.Registry;
8  import java.rmi.registry.LocateRegistry;
9  import java.rmi.RemoteException;
10 import java.rmi.server.UnicastRemoteObject;
11 import java.sql.SQLException;
12 import java.time.LocalDateTime;
13 import java.util.ArrayList;
14 import java.util.logging.Level;
15 import java.util.logging.Logger;
16
17 public class Server extends UnicastRemoteObject implements Hello {
18     private ModeloAlumno modeloAlumno;
19     private ModeloVuelos modeloVuelos;
20
21     public Server() throws RemoteException, SQLException{
22         this.modeloAlumno = new ModeloAlumno();
23         this.modeloVuelos = new ModeloVuelos();
24     }
25
26     @Override
27     public String sayHello(String persona){
28         System.out.println("Hora de peticion: "+LocalDateTime.now().toString()+" : "+persona);
29         return "Hello, world! "+persona;
30     }
31
32     @Override
33     public String sayHello(){
34         System.out.println("Hora de peticion: "+LocalDateTime.now().toString());
35         return "Hello, world! ";
36     }
37
38     @Override
39     public int insertaAlumno(String nombre, String paterno, String materno) throws RemoteException {
40         try {
41             return modeloAlumno.insertaAlumno(nombre, paterno, materno);
42         } catch (SQLException ex) {
43             Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
44             return 0;
45         }
46     }
47
48     @Override
49     public int actualizaAlumno(int idAlumno, String nombre, String paterno, String materno) throws RemoteException {
50         try {
51             return modeloAlumno.actualizaAlumno(idAlumno,nombre, paterno, materno);
52         } catch (SQLException ex) {
53             Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
54             return 0;
55         }
56     }
57
58     @Override
59     public ArrayList<Vuelo> vuelosHistoricos() throws RemoteException {
60         return modeloVuelos.vuelosHistoricos();
61     }
62
```

```
63      @Override
64      public ArrayList<Vuelo> vuelosDisponibles(String fecha) throws RemoteException {
65          try {
66              return modeloVuelos.vuelosDisponibles(fecha);
67          } catch (SQLException ex) {
68              Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
69          }
70          return null;
71      }
72
73      @Override
74      public Vuelo obtenerVuelo(int idVuelo) throws RemoteException{
75          try {
76              return modeloVuelos.obtenVuelo(idVuelo);
77          } catch (SQLException ex) {
78              System.out.println(ex.toString());
79              Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
80          }
81          return null;
82      }
83
84      @Override
85      public ArrayList<Persona> obtenerPersonasVuelo(int idVuelo) {
86          return modeloVuelos.obtenerPersonasVuelo(idVuelo);
87      }
88
89      @Override
90      public ArrayList<Vuelo> vuelosHistoricosPersona(int idPersona) throws RemoteException {
91          try {
92              return modeloVuelos.vuelosHistoricosPersona(idPersona);
93          } catch (SQLException ex) {
94              Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
95          }
96          return null;
97
98      }
99
100     @Override
101     public ArrayList<Vuelo> vuelosDisponiblesPersona(String fecha, int idPersona) throws RemoteException {
102         return this.modeloVuelos.vuelosDisponiblesPersona(fecha, idPersona);
103     }
104
105     @Override
106     public ArrayList<Vuelo> vuelosAnterioresPersona(String fecha, int idPersona) throws RemoteException {
107         return modeloVuelos.vuelosAnterioresPersona(fecha, idPersona);
108     }
109
110     @Override
111     public ArrayList<Lugar> obtenerLugares() throws RemoteException {
112         try {
113             return modeloVuelos.obtenerLugares();
114         } catch (SQLException ex) {
115             Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
116         }
117         return null;
118     }
119
120     @Override
121     public ArrayList<Vuelo> vuelosOrigenDestino(int idOrigen, int idDestino) throws RemoteException {
122         return this.modeloVuelos.vuelosOrigenDestino(idOrigen, idDestino);
123     }
124
125     public static void main(String args[]) {
126         try {
127             Server obj = new Server();
128             Registry registry = LocateRegistry.createRegistry(1010);
129             registry.bind("Hello", obj);
130             System.out.println("Server ready");
```

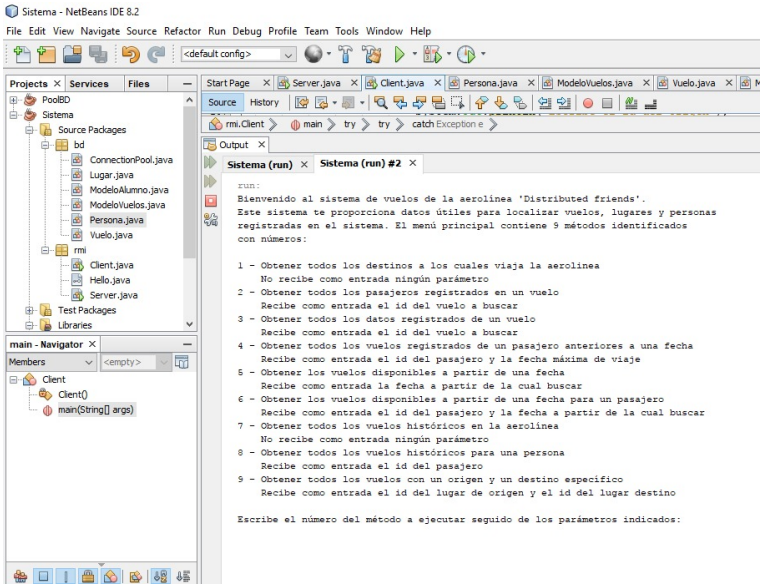
```
131             } catch (Exception e) {
132                 System.err.println("Server exception: " + e.toString());
133                 e.printStackTrace();
134             }
135         }
136     }
```

1.2. Resultados:

Al ejecutar el servidor, es posible que los clientes invoquen los métodos:



De vista al usuario, se mostrará un menú donde se enlistan los métodos que puede invocar, así como los parámetros requeridos por el método:



Al seleccionar un método, el servidor le muestra un mensaje indicándole los parámetros a introducir. Finalmente, devuelve los resultados de acuerdo al método:

```
Escribe el número del método a ejecutar seguido de los parámetros indicados:
1
[Lugar{idLugar=1, nombre=New York}, Lugar{idLugar=2, nombre=Chennai}, Lugar{idLugar=3, nombre=Santorini
BYE =)
BUILD SUCCESSFUL (total time: 43 seconds)
```

Capítulo 2

Intercambio de llaves Diffie Hellman

Para el protocolo de intercambio de llaves Diffie Hellman se establece un secreto compartido entre los participantes. La implementación que se llevo a cabo está basada en la documentación de Oracle [1], donde viene presentado el ejemplo de dos y tres partes involucradas. Por este motivo se tuvo que adecuar para n partes involucradas. La solución propuesta es por medio de una clase de ManejadorLlaves.java, la cual busca gestionar las llaves de los clientes que se conectan para poder encriptar de acuerdo al agreement que se tuvo con cada uno de ellos.

2.1. RMI

Se modificaron las clases existentes, Cliente, Server, y la interfaz Hello, en el paquete RMI del proyecto.

Cliente.java

El archivo Cliente.java se adecuó para generar las llaves del lado del cliente y realizar el agreement con el servidor. Además de esto, se realizó la adecuación de cada uno de los métodos ya que se recibe ahora un arreglo de bytes ya que la información viaja encriptada y s requiere desencriptar los mensajes enviados para cada uno de los métodos.

```
1  int idLlave  = stub.crearLlave();
2  byte[] serverPubKeyEnc = stub.obtenLlave(idLlave);
3
4  LlaveCliente llaveCliente = new LlaveCliente(serverPubKeyEnc);
5  byte[] clientPubKeyEnc = llaveCliente.obtenLlave();
6
```

[1] https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

```
7  stub.coordLlave(idLlave, clientPubKeyEnc);
8  llaveCliente.coordinaConServidor();
```

```
1  case 1: //no necesita mandar parámetros
2      lugares = stub.obtenerLugares(idLlave, clientPubKeyEnc);
3      pars = stub.obtenParametrosDeCifrado(idLlave, clientPubKeyEnc);
4      recovered = llaveCliente.decriptaMensaje(lugares, pars);
5      arregloRes = (ArrayList) deserialize(recovered);
6      System.out.println(arregloRes.toString());
7      break;
8
9  case 2: // si necesita mandar parámetros
10     System.out.println("Escribe el número de vuelo");
11     opcion = Integer.parseInt(reader.readLine());
12     parametroAMandar = llaveCliente.encriptaMensaje(ByteBuffer.allocate(4).putInt(opcion).array());
13
14     //personas = stub.obtenerPersonasVuelo(opcion, idLlave, clientPubKeyEnc);
15     personas = stub.obtenerPersonasVuelo(parametroAMandar, idLlave, clientPubKeyEnc,
16                 llaveCliente.obtenParametrosDeCifrado());
17     pars  = stub.obtenParametrosDeCifrado(idLlave, clientPubKeyEnc);
18     recovered = llaveCliente.decriptaMensaje(personas, pars);
19     arregloRes = (ArrayList) deserialize(recovered);
20     System.out.println(arregloRes.toString());
```

Server.java

El server se adecuó para que cada uno de los métodos propuestos devuelva un objeto tipo *byte[]* ya que es necesario que con la seguridad implementada devuelva el texto cifrado. Esto tiene la implicación que el archivo Hello también se vea modificado. Adicional, se añadieron dos métodos que funcionan para crear la llave y obtener la llave.

```
1      public byte[] converterByte(String objeto) throws RemoteException, SQLException, IOException{
2          ByteArrayOutputStream out = new ByteArrayOutputStream();
3          ObjectOutputStream os = new ObjectOutputStream(out);
4          os.writeObject(objeto);
5          byte[] cleartext = out.toByteArray();
6          return cleartext;
7      }
8
9   public byte[] serializa(Object objeto) {
10         try{
11
12         ByteArrayOutputStream out = new ByteArrayOutputStream();
13         ObjectOutputStream os = new ObjectOutputStream(out);
14         os.writeObject(objeto);
15         byte[] cleartext = out.toByteArray();
16         return cleartext;
17         }catch(Exception e){
18             System.out.println(e.toString());
19             return null;
20         }
21     }
22
23     @Override
24     public int crearLlave() throws RemoteException{
25         //System.out.println("Creando llave");
26         return manejadorLlaves.crearLlave();
27     }
28
29     @Override
30     public byte[] obtenLlave(int llaveId) throws RemoteException{
31         //System.out.println("Regresando llave");
32
33         return manejadorLlaves.obtenerLlave(llaveId);
```

```
34      }
```

Hello.java

Esta interfaz tiene la misma lógica, solamente añadiendo los métodos indicados en el server. De la misma manera, se adecua el tipo de objeto que se devuelve para cada uno de los métodos.

```
1  public byte[] obtenParametrosDeCifrado(int llaveId, byte[] clientPubKeyEnc) throws IOException;
2  public byte[] vuelosHistoricos(int llaveId, byte[] clientPubKeyEnc) throws RemoteException,
3      SQLException, IOException;
4  public byte[] vuelosDisponibles(String fecha, int llaveId, byte[] clientPubKeyEnc) throws RemoteException,
5      SQLException, IOException;
```

2.1.1. Seguridad

Además se creó un nuevo paquete que contiene las clases necesarias para el manejo de seguridad en el envío de mensajes entre el cliente y el servidor.

LlaveCliente.java

Clase que simula el comportamiento de la llave del cliente, implementa los métodos necesarios para encriptar y desencriptar los mensajes, además de los métodos auxiliares para lograrlo:

```
1  public class LlaveCliente {
2      public byte[] llaveInicialServidor;
3      //****
4      private KeyFactory bobKeyFac;
5      private PublicKey alicePubKey;
6      private DHParameterSpec dhParamFromAlicePubKey;
7      private KeyPairGenerator bobKpairGen;
8      private KeyAgreement bobKeyAgree;
9      private KeyPair bobKpair;
10     private SecretKeySpec bobAesKey;
11     private Cipher bobCipher;
12     private AlgorithmParameters aesParams;
13
14     public LlaveCliente(byte[] llaveServidor) throws NoSuchAlgorithmException,
15             InvalidKeySpecException, InvalidAlgorithmParameterException, InvalidKeyException,
16             NoSuchPaddingException{
17
18         this.llaveInicialServidor = llaveServidor;
19
20         // Bob has received Alice's public key in encoded format. He instantiates a
21         // DH public key from the encoded key material.
22         this.bobKeyFac = KeyFactory.getInstance("DH");
23         X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(llaveServidor);
24         this.alicePubKey = bobKeyFac.generatePublic(x509KeySpec);
25         this.dhParamFromAlicePubKey = ((DHPublicKey)this.alicePubKey).getParams();
26
27         // Bob creates his own DH key pair
28         this.bobKpairGen = KeyPairGenerator.getInstance("DH");
29         bobKpairGen.initialize(dhParamFromAlicePubKey);
30         this.bobKpair = bobKpairGen.generateKeyPair();
31
32         // Bob creates and initializes his DH KeyAgreement object
33         this.bobKeyAgree = KeyAgreement.getInstance("DH");
34         this.bobKeyAgree.init(bobKpair.getPrivate());
35
36         // Bob encrypts, using AES in CBC mode
```

```
37          this.aesParams = AlgorithmParameters.getInstance("AES");
38          this.bobCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
39      }
40
41      // Bob encodes his public key, and sends it over to Alice.
42      public byte[] obtenLlave(){
43          return this.bobKpair.getPublic().getEncoded();
44      }
45
46      public void coordinaConServidor() throws InvalidKeyException{
47          // Bob uses Alice's public key for the first (and only) phase
48          // of his version of the DH protocol.
49          System.out.println("BOB: Execute PHASE1 ...");
50          bobKeyAgree.doPhase(this.alicePubKey, true);
51          this.generaLlaveSecreta();
52          this.bobCipher.init(Cipher.ENCRYPT_MODE, this.bobAesKey);
53      }
54
55      public int longitudSecreto(byte[] bobSharedSecret) throws IllegalStateException,
56              ShortBufferException{
57          return bobKeyAgree.generateSecret(bobSharedSecret, 0);
58      }
59
60      private byte[] generarSecreto(){
61          byte[] tmp = bobKeyAgree.generateSecret();
62          for(int i=0; i< tmp.length ; i++) {
63           System.out.print(tmp[i] +" ");
64          }
65          return tmp;
66      }
67
68      private void generaLlaveSecreta(){
69          this.bobAesKey = new SecretKeySpec(this.generarSecreto(), 0, 16, "AES");
70      }
71
72      public byte[] encriptaMensaje(byte[] objetoEnBytes) throws IllegalBlockSizeException,
73              BadPaddingException {
74         return bobCipher.doFinal(objetoEnBytes);
75      }
76
77      public byte[] obtenParametrosDeCifrado() throws IOException{
78          return bobCipher.getParameters().getEncoded();
79      }
80
81      public byte[] decriptaMensaje(byte[] objetoEncriptado, byte[] encodedParams) {
82          try{
83              aesParams.init(encodedParams);
84              bobCipher.init(Cipher.DECRYPT_MODE, bobAesKey, aesParams);
85          return bobCipher.doFinal(objetoEncriptado);
86          }catch(Exception e){
87             System.out.println("Error Llave Cliente:");
88             System.out.println(e.toString());
89          }
90          return null;
91      }
92
93      //Converts a byte to hex digit and writes to the supplied buffer
94      private static void byte2hex(byte b, StringBuffer buf) {
95          char[] hexChars = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
96                  '9', 'A', 'B', 'C', 'D', 'E', 'F' };
97          int high = ((b & 0xf0) >> 4);
98          int low = (b & 0x0f);
99          buf.append(hexChars[high]);
100         buf.append(hexChars[low]);
101     }
102
103     // Converts a byte array to hex string
104     private static String toHexString(byte[] block) {
```

```
105         StringBuffer buf = new StringBuffer();
106         int len = block.length;
107         for (int i = 0; i < len; i++) {
108             byte2hex(block[i], buf);
109             if (i < len-1) {
110                 buf.append(":");
111             }
112         }
113         return buf.toString();
114     }
115 }
```

LlaveServidor.java

Clase que simula el comportamiento de la llave del servidor, implementa los métodos necesarios para encriptar y desencriptar los mensajes, además de los métodos auxiliares para lograrlo:

```
1  package seguridad;
2
3  import java.io.IOException;
4  import java.security.AlgorithmParameters;
5  import java.security.InvalidAlgorithmParameterException;
6  import java.security.InvalidKeyException;
7  import java.security.KeyFactory;
8  import java.security.KeyPair;
9  import java.security.KeyPairGenerator;
10 import java.security.NoSuchAlgorithmException;
11 import java.security.PublicKey;
12 import java.security.spec.InvalidKeySpecException;
13 import java.security.spec.X509EncodedKeySpec;
14 import javax.crypto.BadPaddingException;
15 import javax.crypto.Cipher;
16 import javax.crypto.IllegalBlockSizeException;
17 import javax.crypto.KeyAgreement;
18 import javax.crypto.NoSuchPaddingException;
19 import javax.crypto.spec.SecretKeySpec;
20
21 /**
22  *
23  * @author mcc06
24  */
25 public class LlaveServidor {
26     private int numeroCoordinacion;
27     private byte[] llaveCliente;
28     private String llavePrivada;
29     private String llavePublica;
30     //**********************
31     private KeyPairGenerator aliceKpairGen;
32     private KeyAgreement aliceKeyAgree;
33     private KeyPair aliceKpair;
34     private KeyFactory aliceKeyFac;
35     private PublicKey bobPubKey;
36     private SecretKeySpec aliceAesKey;
37     private AlgorithmParameters aesParams;
38     private Cipher aliceCipher;
39
40     public LlaveServidor(int numeroCoordinacion)
41     throws NoSuchAlgorithmException, InvalidKeyException, NoSuchPaddingException{
42         this.numeroCoordinacion = numeroCoordinacion;
43         this.llavePrivada = "";
44         this.llaveCliente = null;
45
46         /*
47          * Alice creates her own DH key pair with 2048-bit key size
48          */
```

```
49          //System.out.println("ALICE: Generate DH keypair ...");
50          this.aliceKpairGen = KeyPairGenerator.getInstance("DH");
51          this.aliceKpairGen.initialize(2048);
52          this.aliceKpair = aliceKpairGen.generateKeyPair();
53
54          // Alice creates and initializes her DH KeyAgreement object
55          //System.out.println("ALICE: Initialization ...");
56          this.aliceKeyAgree = KeyAgreement.getInstance("DH");
57          this.aliceKeyAgree.init(this.aliceKpair.getPrivate());
58
59          //algoritmo empleado para la encodeada
60          this.aesParams = AlgorithmParameters.getInstance("AES");
61          this.aliceCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
62      }
63
64      public byte[] obtenLlaveInicial(){
65          return aliceKpair.getPublic().getEncoded();
66      }
67
68      public void coordinaConCliente(byte[] bobPubKeyEnc)
69      throws NoSuchAlgorithmException, InvalidKeyException, InvalidKeySpecException{
70          /*
71           * Alice uses Bob's public key for the first (and only) phase
72           * of her version of the DH
73           * protocol.
74           * Before she can do so, she has to instantiate a DH public key
75           * from Bob's encoded key material.
76           */
77          try{
78              this.aliceKeyFac = KeyFactory.getInstance("DH");
79              X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(bobPubKeyEnc);
80              this.bobPubKey = aliceKeyFac.generatePublic(x509KeySpec); // ver a detalle que hace aqui ¿?
81
82
83              //System.out.println("ALICE: Execute PHASE1 ...");
84              this.aliceKeyAgree.doPhase(bobPubKey, true);
85              this.generaLlaveSecreta();
86              this.aliceCipher.init(Cipher.ENCRYPT_MODE, this.aliceAesKey);
87              this.llaveCliente = bobPubKeyEnc;
88          }catch(Exception e){
89              System.out.println("Error en Llave servidor - coordina con cliente");
90              System.out.println(e.toString());
91          }
92
93      }
94
95      private byte[] generarSecreto(){
96          byte [] tmp = aliceKeyAgree.generateSecret();
97          //System.out.println("generate secret llave servidor");
98          for(int i=0; i< tmp.length ; i++) {
99           System.out.print(tmp[i] +" ");
100         }
101         return tmp;
102     }
103
104     private void generaLlaveSecreta(){
105         this.aliceAesKey = new SecretKeySpec(this.generarSecreto(), 0, 16, "AES");
106     }
107
108     public byte[] encriptaMensaje(byte[] objetoEnBytes) throws IllegalBlockSizeException, BadPaddingException{
109
110         try{
111             aliceCipher.init(Cipher.ENCRYPT_MODE, aliceAesKey, aliceCipher.getParameters());
112             byte [] tmp = aliceCipher.doFinal(objetoEnBytes);
113             return tmp;
114         }catch(Exception e){
115             System.out.println("Encripta mensaje");
116             System.out.println(e.toString());
```

```
117         }
118         return null;
119     }
120 
121     public byte[] decriptaMensaje(byte[] objetoEncriptado, byte[] encodedParams) {
122 
123         try{
124             aesParams.init(encodedParams);
125             aliceCipher.init(Cipher.DECRYPT_MODE, aliceAesKey, aesParams);
126         return aliceCipher.doFinal(objetoEncriptado);
127         }catch(Exception e){
128            System.out.println("Error Llave Cliente:");
129            System.out.println(e.toString());
130         }
131         return null;
132 
133     }
134 
135 
136 
137     public byte[] obtenParametrosDeCifrado() throws IOException{
138         return aliceCipher.getParameters().getEncoded();
139     }
140 
141     public void guardaLlavePublicaCliente(byte[] llaveCliente){
142         this.llaveCliente = llaveCliente;
143     }
144 
145     public byte[] getLlavePublicaCliente(){
146         return this.llaveCliente;
147     }
148 
149     /*
150      * Converts a byte to hex digit and writes to the supplied buffer
151      */
152     private static void byte2hex(byte b, StringBuffer buf) {
153         char[] hexChars = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
154                 '9', 'A', 'B', 'C', 'D', 'E', 'F' };
155         int high = ((b & 0xf0) >> 4);
156         int low = (b & 0x0f);
157         buf.append(hexChars[high]);
158         buf.append(hexChars[low]);
159     }
160 
161     /*
162      * Converts a byte array to hex string
163      */
164     public static String toHexString(byte[] block) {
165         StringBuffer buf = new StringBuffer();
166         int len = block.length;
167         for (int i = 0; i < len; i++) {
168             byte2hex(block[i], buf);
169             if (i < len-1) {
170                 buf.append(":");
171             }
172         }
173         return buf.toString();
174     }
175 }
```

ManejadorLlaves.java

Al tener ahora distintos clientes que pueden interactuar con el servidor, es necesario contar con un manejador que gestione las llaves. De esta manera, a través de un id, lograremos identificar que llaves se ocupan con cada cliente.

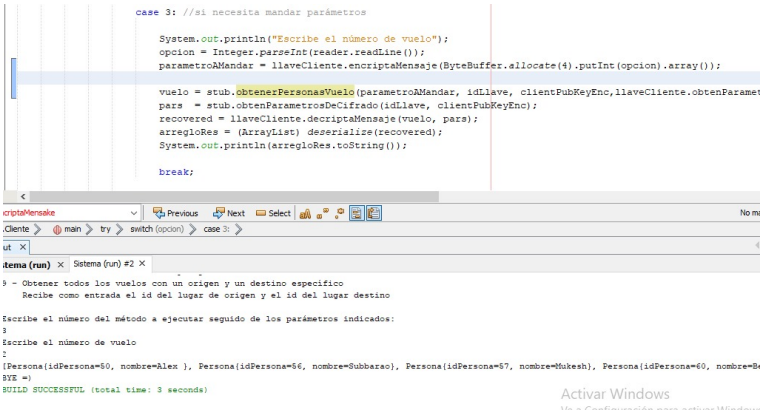
```java
package seguridad;

import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

/**
 *
 * @author mcc06
 */
public class ManejadorLlaves {
    private ArrayList<LlaveServidor> manejadorLlaves;
    private int totalCreados;

    public ManejadorLlaves(){
        this.manejadorLlaves = new ArrayList<LlaveServidor>();
        this.totalCreados = 0;
        this.manejadorLlaves.add(0,null);
    }

    /**
     * Este método se emplea para crear un nuevo objeto de llave privada,
     * de forma que para cada nuevo cliente sea lo primero que se hace y puedan
     * llevarse a cabo los siguientes pasos de coordinación
     * @return indice del arraylist donde se encuentra la llave indicada
     */
    public int crearLlave(){
        this.totalCreados++;
        try{
            LlaveServidor ll = new LlaveServidor(this.totalCreados);
            this.manejadorLlaves.add(this.totalCreados,ll);
            return this.totalCreados;

        }catch(Exception e){
            System.out.println(e.toString());
            return 0;
        }
    }

    public byte[] obtenerLlave(int idLlaveCliente){
        if(this.manejadorLlaves.get(idLlaveCliente)!=null){
            return this.manejadorLlaves.get(idLlaveCliente).obtenLlaveInicial();
        }else{
            return null;
        }
    }

    public byte[] obtenParametrosDeCifrado(int idLlaveCliente) throws IOException{
        if(this.manejadorLlaves.get(idLlaveCliente)!=null){
            return this.manejadorLlaves.get(idLlaveCliente).obtenParametrosDeCifrado();
        }else{
            return null;
        }
    }

    public boolean coordLlave(int idLlaveCliente, byte[] llavePublicaCliente){
        if(this.manejadorLlaves.get(idLlaveCliente)!=null){
            try {
                this.manejadorLlaves.get(idLlaveCliente).coordinaConCliente(llavePublicaCliente);
                return true;
```

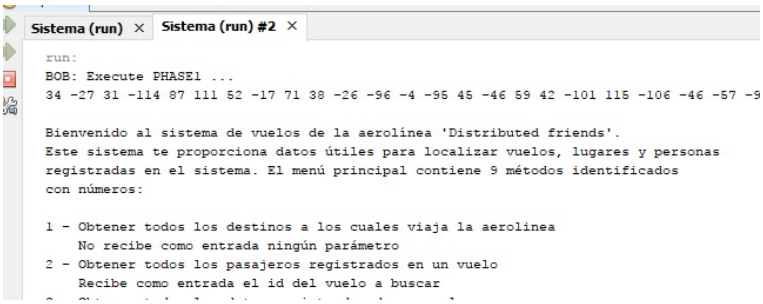
```
69              } catch (Exception ex) {
70                  System.out.println("Error al coordinar la llave del cliente: "+idLlaveCliente);
71              }
72          }
73          return false;
74      }
75
76      public byte[] encripta(byte[] llavePublicaCliente, byte[] cosa)
77      throws IllegalBlockSizeException, BadPaddingException{
78          //System.out.println("encriptando en el servidor con la llave");
79          return encuentraLlave(llavePublicaCliente).encriptaMensaje(cosa);
80      }
81
82      public byte[] desencripta(byte[] llavePublicaCliente, byte[] cosa, byte[] paramsEncriptadoCliente)
83      throws IllegalBlockSizeException, BadPaddingException{
84          //System.out.println("encriptando en el servidor con la llave");
85          return encuentraLlave(llavePublicaCliente).decriptaMensaje(cosa, paramsEncriptadoCliente);
86      }
87
88      private LlaveServidor encuentraLlave(byte[] llavePublica){
89
90          LlaveServidor temp = null;
91          int indice = 1;
92          boolean encontrado = false;
93          while(indice < this.manejadorLlaves.size() && !encontrado){
94              temp = this.manejadorLlaves.get(indice);
95
96              if(Arrays.equals(temp.getLlavePublicaCliente(), llavePublica)){
97                  encontrado=true;
98                  System.out.println("Llave encontrada es la: "+indice);
99              }
100             indice++;
101         }
102         System.out.println(temp.toString());
103         return temp;
104     }
105
106
107 }
```

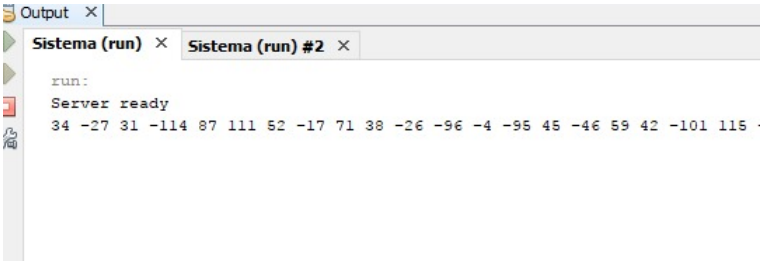
2.2. Resultados

En la siguiente imagen se muestra una consulta del cliente y la respuesta del servidor. Hay que notar que el parámetro que se envía está encriptado y de la misma manera lo que se recibe tiene que pasar por un proceso de desencripción y des-serialización:

Al realizar la coordinación, imprimimos los shared secrets para el servidor y el cliente. En la primera imagen se muestra el shared secret del Cliente, mientras que el segundo se muestra el shared secret del Server.


case 3: //si necesita mandar parámetros
System.out.println("Escribe el número de vuelo");
opcion = Integer.parseInt(reader.readLine());
parametroAMandar = llaveCliente.encriptaMensaje(ByteBuffer.allocate(4).putInt(opcion).array());
vuelo = stub.obtenerPersonasVuelo(parametroAMandar, idLlave, clientPubKeyEnc,llaveCliente.obtenParamet
pars = stub.obtenParametrosDeCifrado(idLlave, clientPubKeyEnc);
recovered = llaveCliente.decriptaMensaje(vuelo, pars);
arregloRes = (ArrayList) deserialize(recovered);
System.out.println(arregloRes.toString());
break;
Previous
Next
Select
main
try
switch (opcion)
case 3:
Sistema (run)
Sistema (run) #2
9 - Obtener todos los vuelos con un origen y un destino específico
Recibe como entrada el id del lugar de origen y el id del lugar destino
Escribe el número del método a ejecutar seguido de los parámetros indicados:
3
Escribe el número de vuelo
2
[Persona{idPersona=50, nombre=Alex }, Persona{idPersona=56, nombre=Subbarao}, Persona{idPersona=57, nombre=Mukesh}, Persona{idPersona=60, nombre=B
BYE =)
BUILD SUCCESSFUL (total time: 3 seconds)
Activar Windows


Sistema (run)
Sistema (run) #2
run:
BOB: Execute PHASE1 ...
34 -27 31 -114 87 111 52 -17 71 38 -26 -96 -4 -95 45 -46 59 42 -101 115 -106 -46 -57 -9
Bienvenido al sistema de vuelos de la aerolínea 'Distributed friends'.
Este sistema te proporciona datos útiles para localizar vuelos, lugares y personas
registradas en el sistema. El menú principal contiene 9 métodos identificados
con números:
1 - Obtener todos los destinos a los cuales viaja la aerolinea
No recibe como entrada ningún parámetro
2 - Obtener todos los pasajeros registrados en un vuelo
Recibe como entrada el id del vuelo a buscar


Output
Sistema (run)
Sistema (run) #2
run:
Server ready
34 -27 31 -114 87 111 52 -17 71 38 -26 -96 -4 -95 45 -46 59 42 -101 115