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

RMI: 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()
24      throws RemoteException; //M
25      public ArrayList<Vuelo> vuelosDisponibles(String fecha)
26      throws RemoteException; //S
27      public Vuelo obtenerVuelo(int idVuelo)
28      throws RemoteException; //F +
29      public ArrayList<Persona> obtenerPersonasVuelo(int idVuelo)
30      throws RemoteException; //M
31      public ArrayList<Vuelo> vuelosHistoricosPersona(int idPersona)
32      throws RemoteException; //S
33      public ArrayList<Vuelo> vuelosDisponiblesPersona(String fecha, int idPersona)
34      throws RemoteException; //F +
35      public ArrayList<Vuelo> vuelosAnteriores Persona(String fecha, int idPersona)
36      throws RemoteException; //M
37      public ArrayList<Lugar> obtenerLugares()
38      throws RemoteException; //S
39      public ArrayList<Vuelo> vuelosOrigenDestino(int idOrigen, int idDestino)
40      throws RemoteException; //F
41
42  }
```

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

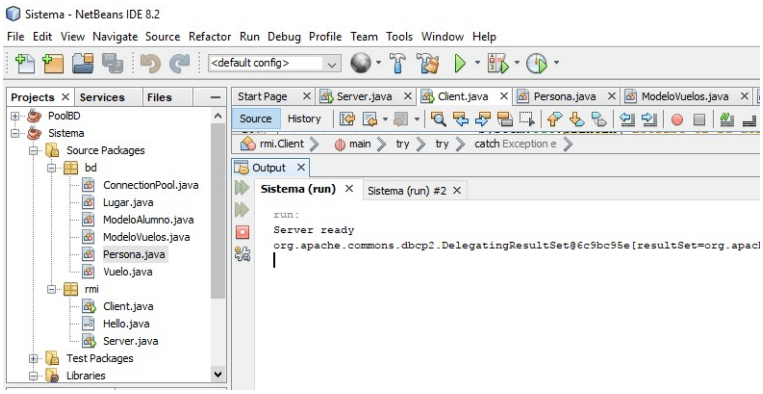
```
1  package rmi;
2  import bd.Lugar;
3  import bd.ModeloAlumno;
4  import bd.ModeloVuelos;
5  import bd.Persona;
6  import bd.Vuelo;
7  import java.rmi.registry.Registry;
8  import java.rmi.registry.LocateRegistry;
9  import java.rmi.RemoteException;
10 import java.rmi.server.UnicastRemoteObject;
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");
```

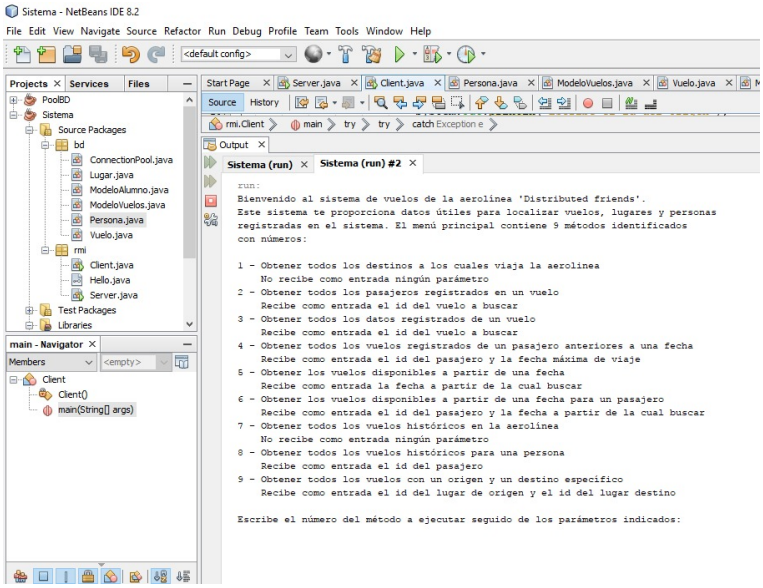
```
131             } catch (Exception e) {
132                 System.err.println("Server exception: " + e.toString());
133                 e.printStackTrace();
134             }
135         }
136     }
```

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)
```