Proceso Básico de Desarrollo con RMI

Práctica 1. (Será realizado en la Sala de Computo)

El objetivo de esta práctica es conocer el proceso básico de funcionamiento de RMI. En este ejercicio se requiere implementar una aplicación distribuida basada en Java RMI, la cual permita a un usuario registrar un usuario y consultar un usuario registrado. Lo datos a registrar corresponden a: nombre del paciente, apellido del paciente, número de la habitación del paciente y edad del paciente. Estas operaciones serán controladas por el cliente mediante un Menú, tal como se muestra la Figura 1.

========== Menú ==========

1. *Registrar paciente*
2. *Consultar paciente*
3. *Salir*

========== ==========

Digite opción:

Figura 1. Menú del cliente

La primera opción permite registrar todos los datos de un paciente. En el lado del servidor los pacientes serán almacenados en un arrrayList. La máxima cantidad de pacientes a registrar será 5.

En la siguiente figura se muestra un diagrama de red con los métodos que puede invocar el cliente de objetos.

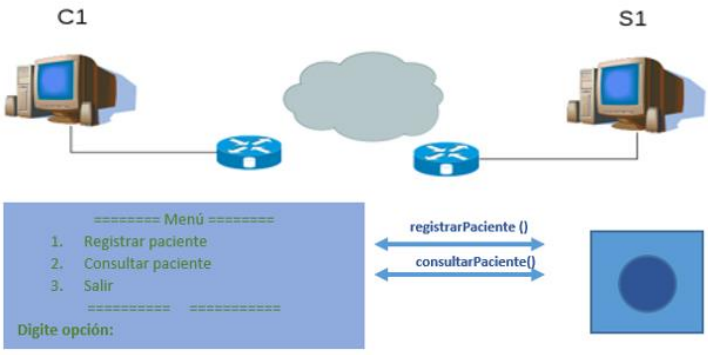

Figura 1. Diagrama de red

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato rar y ser enviada a la plataforma. El nombre del archivo comprimido debe seguir el siguiente formato lsd_rmi_p1_apellidoN1_apellidoN2.rar. Donde apellidoN1 corresponde al primer apellido del primer integrante y apellidoN2 corresponde al primer apellido del segundo integrante del grupo. En la carpeta se deben especificar el rol de cada integrante.

Antes de iniciar, estructurar una carpeta de trabajo donde se ubicaran los archivos fuente (directorio 'src') y los archivos binarios (bytecode) (directorio bin). El nombre del directorio de trabajo debe coincidir con el nombre del archivo comprimido. Usar el archivo dir_repaso.bat para crear la estructura mostrada en la Figura 2.

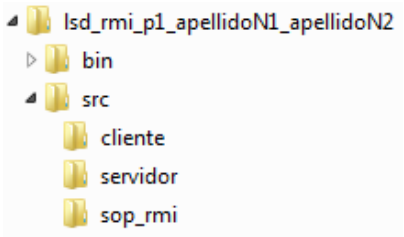

Figura 2. Estructura de directorios

1. Diseñar e implementar los componentes de la interface

Los archivos GestionUsuariosInt.java y ClsGestionUsuariosImpl.java deben estar ubicados en la carpeta sop_rmi.

1. a) Definiendo la interface remota: Editar el archivo GestionUsuariosInt.java y declarar 2 métodos que podrán ser invocados por el proceso cliente, como se muestran en la figura 3.

```
package sop_rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

//Hereda de la clase Remote, lo cual la convierte en interfaz remota
public interface GestionUsuariosInt extends Remote{

    public boolean registrarUsuario(ClsUsuarioDTO objUsuario) throws RemoteException;
    public ClsUsuarioDTO consultarUsuario(int numHabitacion) throws RemoteException;

}
```

Figura 3. Interface GestionUsuariosInt.*java*

1.b) Implementando la Interface Remota: Editar el archivo ClsGestionUsuariosImpl.java correspondiente al código del objeto servidor. En el archivo implementar los métodos de la interfaz remota GestionUsuariosInt.java. En la implementación de la clase ClsUsuarisosProyectosImpl, esta clase hereda de la clase UnicastRemoteObject e implementa la interface GestionUsuariosInt como se muestra en la figura 4. También es necesario que los métodos implementados en la clase puedan lanzar la excepción RemoteException.

```
package sop_rmi;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class ClsGestionUsuarios extends UnicastRemoteObject implements GestionUsuariosInt
{
    private ArrayList<ClsUsuarioDTO> usuarios;

    public ClsGestionUsuarios() throws RemoteException
    {
        super(); //invoca al constructor de la clase base
        usuarios= new ArrayList();
    }

    @Override
    public boolean registrarUsuario(ClsUsuarioDTO objUsuario) throws RemoteException {
        return this.usuarios.add(objUsuario);
    }

    @Override
    public ClsUsuarioDTO consultarUsuario(int numHabitacion) throws RemoteException {
         return this.usuarios.get(0);
    }
}
```

Figura 4. Declaración de la clase ClsGestionUsuariosImpl.*java*

1.c) Implementar la clase que encapsula los datos del usuarios: Editar un archivo denominado ClsUsuarioDTO, asignar los atributos del usuario, crear el constructor y los métodos get. Como se muestra a continuación:

```
package sop_rmi;

import java.io.Serializable;

public class ClsUsuarioDTO implements Serializable
{
        private int numHabitacion;
        private String nombres;
        private String apellidos;
        private int edad;

    public ClsUsuarioDTO(int numHabitacion, String nombres, String apellidos, int edad) {...6 lines }

    public int getNumHabitacion() {...3 lines }

    public void setNumHabitacion(int numHabitacion) {...3 lines }

    public String getNombres() {...3 lines }

    public void setNombres(String nombres) {...3 lines }

    public String getApellidos() {...3 lines }

    public void setApellidos(String apellidos) {...3 lines }

    public int getEdad() {...3 lines }

    public void setEdad(int edad) {...3 lines }

}
```

2. Crear el código del cliente y generar los stubs

Tener en cuenta que: Para compilar los archivos fuente se debe ubicar en el directorio 'src'.

2.a Escribir el código del cliente de objetos: Editar un archivo denominado *ClienteDeObjetos.java*, (basarse en el código fuente ejemplo_codigo_explicado_clase.rar), e incluir el código correspondiente que permita implementar las actividades indicadas en el enunciado del problema. El archivo ClienteDeObjetos.java debe ubicarse en la carpeta cliente.

En el código del cliente se debe obtener una referencia remota (es decir, una referencia que corresponda a un objeto remoto) asociada al servicio para luego simplemente invocar de forma convencional sus métodos, aunque teniendo en cuenta que pueden generar la excepción RemoteException. Un ejemplo de cómo puede ser obtenida la referencia remota del objeto se puede ver en la Figura 5, y el método que permite consultar la referencia remota al rmiregistry es mostrado en la figura 6. El nombre del servicio debe ser **ObjetoRemotoUsuario.**

```
public class ClienteDeObjetos
{

    private static GestionUsuariosInt objRemoto;

    public static void main(String[] args)
    {
        int numPuertoRMIRegistry = 0;
        String direccionIpRMIRegistry = "";

        System.out.println("Cual es el la dirección ip donde se encuentra  el rmiregistry ");
        direccionIpRMIRegistry = cliente.UtilidadesConsola.leerCadena();
        System.out.println("Cual es el número de puerto por el cual escucha el rmiregistry ");
        numPuertoRMIRegistry = cliente.UtilidadesConsola.leerEntero();

        objRemoto = (GestionUsuariosInt) UtilidadesRegistroC.obtenerObjRemoto(direccionIpRMIRegistry,numPuertoRMIRegistry, "ObjetoRemotoUsuario");
        MenuPrincipal();

    }
```

Figura 5. Ejemplo para obtener del rmiregistry una referencia remota del servicio.

```
public class UtilidadesRegistroC
{

    public static Remote obtenerObjRemoto(int puerto, String dirIP, String nameObjReg)
    {
        String URLRegistro;
        URLRegistro  = "rmi://" + dirIP + ":" + puerto + "/"+nameObjReg;
        try
        {
            return Naming.lookup(URLRegistro);
        }
        catch (NotBoundException | MalformedURLException | RemoteException e)
        {
            System.out.println("Excepcion en obtencion del objeto remoto"+ e);
            return null;
        }
    }

}
```

Figura 6. Ejemplo del método que permite obtener del rmiregistry una referencia remota del servicio

En la figura 6 el programa usa el método estático lookup para obtener del rmiregistry una referencia remota del servicio. Los argumentos de entrada del método main corresponden a: el nombre de la maquina donde se encuentra el proceso rmiregistry y el puerto donde está escuchando el proceso rmiregistry.

Observe el uso de cast para adaptar la referencia devuelta por lookup, que corresponde a la interfaz Remote, al tipo de interfaz concreto (GestionusuariosInt). Una vez obtenida la referencia remota, la invocación del método es convencional, requiriendo el tratamiento de las excepciones que

puede generar.

2.b. Generando Stubs

- Compilar el código fuente del objeto de implementación (*ClsGestionUsuarioss.java*) con la herramienta *javac* (Compilador java).

```
javac -d ../bin sop_rmi/*.java
```

3. Hacer accesibles las clases a través de la red.

3.a. Implementar el servidor: Editar un archivo denominado *ServidorDeObjetos.java*, basarse en el código fuente ejemplo_codigo_explicado_clase.rar e incluir el código correspondiente que permita instanciar el objeto servidor y registrarlo en el N_S, un ejemplo de cómo puede ser registrado un objeto en el n_s puede observarse en la figura 7, en la cual los argumentos de entrada del método main corresponden a: el nombre de la maquina donde se encuentra el proceso rmiregistry y el puerto donde está escuchando el proceso rmiregistry. El método mostrado en la figura 7 se apoya de los métodos que permiten crear una referencia remota al n_s, el cual es mostrado en la figura 8, y registrar un objeto remoto en el n_s como es el mostrado en la figura 9. El nombre del objeto remoto debe ser **ObjetoRemotoUsuario**.

```
package servidor;
import java.rmi.RemoteException;
import sop_rmi.ClsGestionUsuarios;

public class ServidorDeObjetos
{
    public static void main(String args[]) throws RemoteException
    {
        int numPuertoRMIRegistry = 0;
        String direccionIpRMIRegistry = "";

        System.out.println("Cual es el la dirección ip donde se encuentra  el rmiregistry ");
        direccionIpRMIRegistry = UtilidadesConsola.leerCadena();
        System.out.println("Cual es el número de puerto por el cual escucha el rmiregistry ");
        numPuertoRMIRegistry = UtilidadesConsola.leerEntero();

        ClsGestionUsuarios objRemoto = new ClsGestionUsuarios();

        try
        {
           UtilidadesRegistroS.arrancarNS(numPuertoRMIRegistry);
           UtilidadesRegistroS.RegistrarObjetoRemoto(objRemoto, direccionIpRMIRegistry, numPuertoRMIRegistry, "ObjetoRemotoUsuario");

        } catch (Exception e)
        {
            System.err.println("No fue posible Arrancar el NS o Registrar el objeto remoto" +  e.getMessage());
        }
    }
}
```

Figura 7. Implementación de la clase ServidorDeObjetos que permite instanciar un objeto de la clase ClsGestionUsuarios y registrarlo en el n_s.

```
public static void arrancarNS(int numPuertoRMI) throws RemoteException
{
        try
        {

            Registry registro = LocateRegistry.getRegistry(numPuertoRMI);
            String[] nombresLigados= registro.list();

            System.out.println("El registro se ha obtenido y se encuentra escuchando en el puerto: " + numPuertoRMI);
            System.out.println("Nombres registrados");
            for(String nombreRegistrado: nombresLigados)
            {
                System.out.println("nombre: " + nombreRegistrado);
            }
        }
        catch(RemoteException e)
        {
            System.out.println("El registro RMI no se localizó en el puerto: " + numPuertoRMI);

            Registry registro = LocateRegistry.createRegistry(numPuertoRMI);
            System.out.println("El registro se ha creado en el puerto: " + numPuertoRMI);
        }

}
```

Figura 8. Método que permite crear una referencia a un n_s que este activo, o crear un n_s si no hay ninguno activo.

```
public static void RegistrarObjetoRemoto(Remote objetoRemoto, String dirIP, int numPuerto, String nombreObjeto)
{
        String UrlRegistro = "rmi://"+dirIP+":"+numPuerto+"/"+nombreObjeto;
        try
        {
            Naming.rebind(UrlRegistro, objetoRemoto);
            System.out.println("Se realizo el registro con la direccion: " +UrlRegistro);
            System.out.println("Esperando peticiones ...");
        } catch (RemoteException e)
        {
            System.out.println("Error en el registro del objeto remoto");
            e.printStackTrace();
        } catch (MalformedURLException e)
        {
            System.out.println("Error url inválida");
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

}
```

Figura 9. Método que permite registrar un objeto remoto en el n_s.

La parte principal de este programa descrita en la clase ServidorDeObjetos, está incluida en la sentencia try y consiste en crear un objeto de la clase que implementa el servicio remoto y darle de alta en el rmiregistry usando el método estático rebind que permite especificar la operación usando un formato de tipo URL.

3 b. Compilar las clases del servidor y cliente

- Compilar las clases del Servidor de Objetos y el Cliente de Objetos con la herramienta javac.

Ubicados en el directorio 'src' el comando sería,
Para compilar el servidor de objetos:

```
javac -d ../bin servidor/*.java
```

Para compilar el cliente de objetos:

```
javac -d ../bin cliente/*.java
```

3.c. Iniciar el registro.

El rmiregistry se lanza desde código.

4. Iniciar la aplicación.

Tener en cuenta que: Para ejecutar la aplicación ubicarse en el directorio 'bin'.

a. Ejecutar el servidor:
Comando general:

```
java Nombre_clase_servidor_obj
```

Por ejemplo, si la clase pertenece al paquete 'servidor':

```
java servidor.ServidorDeObjetos
```

b. Ejecutar el cliente:
Comando general:

```
java Nombre_clase_cliente_obj
```

Por ejemplo, si la clase pertenece al paquete 'cliente':

```
java cliente.ClienteDeObjetos
```

Todos los archivos entregados deben estar ubicados en las carpetas cliente y servidor respectivamente. Los archivos fuentes no deben estar vinculados con ningún IDE de desarrollo. El servidor de objetos y el cliente de objetos deben recibir como parámetro el nombre del equipo o dirección ip y puerto donde se encuentra ejecutándose el proceso rmiregistry.