Paso de parámetros en RMI

Práctica 2. (Será realizado en la Sala de Computo)

El objetivo de esta práctica es conocer el proceso de paso de párametros en RMI. En este ejercicio se requiere implementar una aplicación distribuida basada en Java RMI, la cual permita a un usuario registrar un paciente, consultar los datos de un paciente registrado en una habitación particular (en el *Servidor de Alertas*) y reportar el registro a un *Servidor de Notificaciones*. Los 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*

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

Figura 1. Menú del cliente

La primera opción permite registrar todos los datos de un paciente. En el lado del *Servidor de Alertas*, los pacientes serán almacenados en un arrrayList. Después de registrar un paciente debe enviarse al *Servidor de Notificaciones* una notificación del éxito o fracaso del registro de un paciente. La máxima cantidad de pacientes a registrar será 5.
En las Figuras 2 y 3, se muestran ejemplos de los mensajes de notificación de éxito y fracaso que se deben mostrar en el Servidor de Notificaciones.

```
======== Notificación ==========
Se registro el paciente NNN
en la habitación ## EXITOSAMENTE
================================
```

Figura 2. Mensaje exitoso.

```
======== Notificación ==========
FALLO el registro del paciente NNN
en la habitación ##
================================
```

Figura 3. Mensaje de fracaso.

En la siguiente figura 4, se muestra un diagrama de red con los métodos que puede invocar el cliente de objetos.

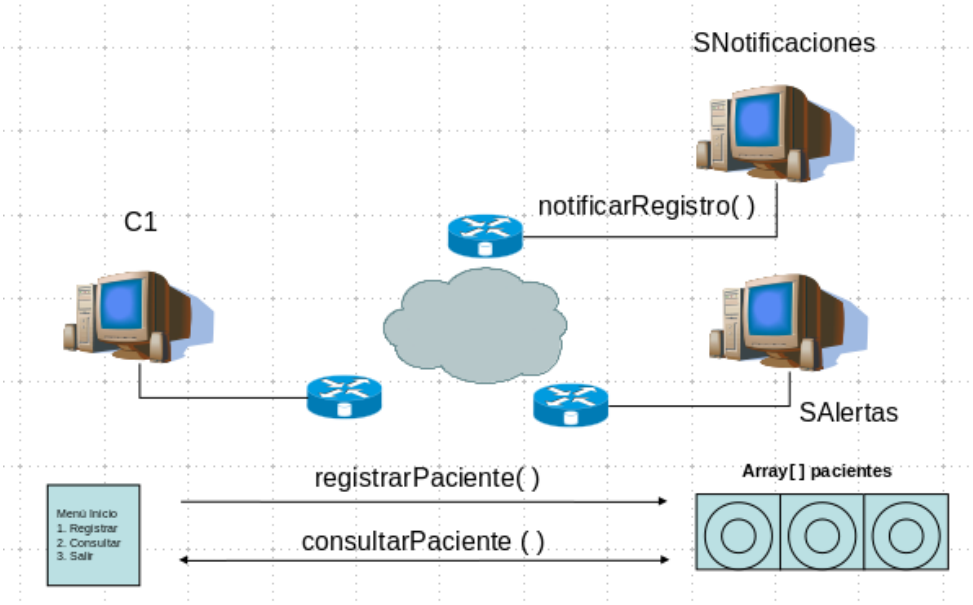

Figura 4 Diagrama de contexto

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato rar o zip y ser enviada a la plataforma. El nombre del archivo comprimido debe seguir el siguiente formato lsd_rmi_p2_apellidoN1_apellidoN2.rar. Donde apellidoN1 corresponde al primer apellido del primer integrante y apellidoN2 corresponde al primer apellido del segundo integrante del grupo. Se debe incluir un archivo *roles.txt* donde se debe especificar el role 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 5.

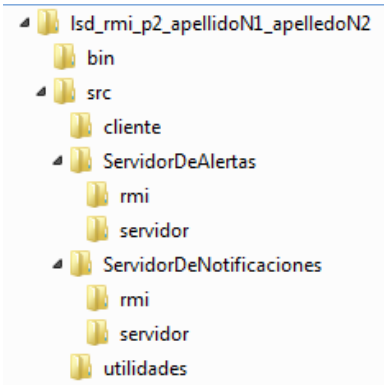

Figura 5. Estructura de directorios

1. Diseñar e implementar los componentes de la interface

1. a) Definiendo las interfaces remotas:

En el Servidor de Alertas:

Los archivos *GestionUsuariosInt.java*, *ClsGestionUsuarios.java* y *ClsUsuarioDTO.java* deben estar ubicados en la carpeta *servidorDeAlertas.rmi*.

Editar el archivo *GestionUsuariosInt.java* y declarar 2 métodos que podrán ser invocados por el proceso cliente, como se muestran en la figura 6.

```
package servidorDeAlertas.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 6. Interface GestionUsuariosInt.java

En el Servidor de Notificación:

Los archivos *NotificacionInt.java*, *ClsNotificacion.java* y *ClsMensajeDTO.java* deben estar ubicados en la carpeta *servidorDeNotificaciones.rmi* de este servidor.

Editar el archivo *NotificaciónInt.java* y declarar 1 método que podrán ser invocados por el proceso servidor de alertas, como se muestran en la figura 7.

```
package servidorDeNotificaciones.rmi;

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

public interface NotificacionInt extends Remote {
    public void notificarRegistro(ClsMensajeNotificacionDTO objNotificacion) throws RemoteException;
}
```

Figura 7. Interface NotificacionInt.java

1.b) Implementando las Interfaces Remotas:

En el Servidor de Alertas:

Editar el archivo *ClsGestionUsuarios.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 *ClsGestionUsuarios*, se hereda de la clase UnicastRemoteObject e

implementa la interface GestionUsuariosInt como se muestra en la figura 8. También es necesario que los métodos implementados en la clase puedan lanzar la excepción RemoteException. Tener en cuenta que el atributo *objReferenciaRemotaNotificacion* será utilizado para almacenar la referencia remota del *Servidor de Notificaciones*, que permitirá enviar la notificación desde el método el método *registrarUsuario()*.

```
package servidorDeAlertas.rmi;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import servidorDeNotificaciones.rmi.ClsMensajeNotificacionDTO;
import servidorDeNotificaciones.rmi.NotificacionInt;
import utilidades.UtilidadesRegistroC;

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

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

    @Override
    public boolean registrarUsuario(ClsUsuarioDTO objUsuario) throws RemoteException {

    @Override
    public ClsUsuarioDTO consultarUsuario(int numHabitacion) throws RemoteException {

    public void consultarReferenciaRemotaDeNotificacion(String direccionIpRMIRegistry, int  numPuertoRMIRegistry)
    {
        this.objReferenciaRemotaNotificacion= (NotificacionInt) UtilidadesRegistroC.obtenerObjRemoto(direccionIpRMIRegistry,
        numPuertoRMIRegistry, "ObjetoRemotoNotificaciones");

    }

}
```

Figura 8. Declaración de la clase *ClsGestionUsuarios.java*

En el Servidor de Notificaciones:

Editar el archivo ClsNotificacion.java correspondiente al código del objeto servidor. En el archivo implementar el método de la interfaz remota NotificacionInt.java. En la implementación de la clase ClsNotificacion, esta clase hereda de la clase UnicastRemoteObject e implementa la interface NotificacionInt como se muestra en la figura 9. También es necesario que los métodos implementados en la clase puedan lanzar la excepción RemoteException.

```
package servidorDeNotificaciones.rmi;

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

public class ClsNotificacion extends UnicastRemoteObject implements NotificacionInt{

    public ClsNotificacion() throws RemoteException
    {
        super();
    }

    @Override
    public void notificarRegistro(ClsMensajeNotificacionDTO objNotificacion) throws RemoteException{
```

Figura 9. Declaración de la clase *ClsNotificacion.java*

1.c) Implementar la clase que encapsula los datos de los usuarios:

En el Servidor de Alertas:

Editar un archivo denominado *ClsUsuarioDTO*, asignar los atributos del usuario, crear el constructor y los métodos get. Como se muestra a continuación:

```
package servidorDeAlertas.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) {
        this.numHabitacion = numHabitacion;
        this.nombres = nombres;
        this.apellidos = apellidos;
        this.edad = edad;
    }

    public int getNumHabitacion() {
        return numHabitacion;
    }
```

Figura 10. Declaración de la clase *ClsUsuarioDTO.java*

En el servidor de Notificaciones:

Editar un archivo denominado *ClsMensajeNotificacionDTO*, asignar los atributos de la notificación, crear el constructor y los métodos get. Como se muestra a continuación:

```
package servidorDeNotificaciones.rmi;

import java.io.Serializable;

public class ClsMensajeNotificacionDTO implements Serializable{
    private int noHabitacion;
    private String nombres;
    private boolean estado;

    public ClsMensajeNotificacionDTO(int noHabitacion, String nombres, boolean estado) {
        this.noHabitacion = noHabitacion;
        this.nombres = nombres;
        this.estado = estado;
    }



    public int getNoHabitacion() {
```

Figura 11. Declaración de la clase *ClsMensajeNotificacionDTO.java*

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 12, y el método que permite consultar la referencia remota al rmiregistry es mostrado en la figura 13. El nombre del servicio debe ser **ObjetoAlmacerUsuario** para el caso del ***Servidor de Alertas*** y **ObjetoRemotoNotificaciones** para el caso del ***Servidor de Notificaciones*****.**

```
public class ClienteDeObjetos
{

    private static GestionUsuariosInt objRemoto;

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

        System.out.println("Cual es el la dirección ip donde se encuentra  el n_s de alertas ");
        direccionIpNS = UtilidadesConsola.leerCadena();
        System.out.println("Cual es el número de puerto por el cual escucha el n_s de alertas ");
        numPuertoNS = UtilidadesConsola.leerEntero();

        objRemoto = (GestionUsuariosInt) UtilidadesRegistroC.obtenerObjRemoto(direccionIpNS,numPuertoNS, "ObjetoAlmacenarUsuario");
        MenuPrincipal();

    }
```

Figura 12. 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 13. Ejemplo del método que permite obtener del rmiregistry una referencia remota del servicio

En la figura 13 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

En el Servidor de Alertas:
- Compilar el código fuente del objeto de implementación (*ClsGestionUsuarios.java*) con la herramienta *javac* (Compilador java).

```
javac -d ../bin ServidorDeAlertas.rmi/*.java
```

En el Servidor de Notificaciones:
- Compilar el código fuente del objeto de implementación (*ClsNotificacion.java*) con la herramienta *javac* (Compilador java).

```
javac -d ../bin ServidorDeNotificaciones.rmi/*.java
```

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

3.a. Implementar el servidor:
En el Servidor de Alertas:
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 remoto y registrarlo en el N_S, un ejemplo de cómo puede ser registrado un objeto en el n_s puede observarse en la figura 14, 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 14 se apoya de los métodos que permiten crear una referencia remota al n_s, el cual es mostrado en la figura 15, y registrar un objeto remoto en el n_s como es el mostrado en la figura 16. El nombre del objeto remoto debe ser **ObjetoAlmacerUsuario**.

```
package servidorDeAlertas.servidor;
import utilidades.UtilidadesConsola;
import utilidades.UtilidadesRegistroS;
import java.rmi.RemoteException;
import servidorDeAlertas.rmi.ClsGestionUsuarios;

public class ServidorDeObjetos
{
    public static void main(String args[]) throws RemoteException
    {
        int numPuertoNS;
        String direccionIpNS;

        System.out.println("Cual es el la dirección ip donde se encuentra  el n_s de alertas");
        direccionIpNS = UtilidadesConsola.leerCadena();
        System.out.println("Cual es el número de puerto por el cual escucha el n_s de alertas");
        numPuertoNS = UtilidadesConsola.leerEntero();


        ClsGestionUsuarios objRemoto = new ClsGestionUsuarios();
        objRemoto.consultarReferenciaRemotaDeNotificacion(direccionIpNS, numPuertoNS);

        try
        {

           UtilidadesRegistroS.RegistrarObjetoRemoto(objRemoto, direccionIpNS, numPuertoNS, "ObjetoAlmacenarUsuario");

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

Figura 14. 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 15. 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 16. 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.

En el Servidor de Notificacioness:
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 remoto y registrarlo en el N_S, un ejemplo de cómo puede ser registrado un objeto en el n_s puede observarse en la figura 17.

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

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

        ClsNotificacion objRemoto = new ClsNotificacion();

        try
        {

           UtilidadesRegistroS.RegistrarObjetoRemoto(objRemoto, direccionNS, numPuertoNS, "ObjetoRemotoNotificaciones");

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

Figura 17. Implementación de la clase ServidorDeObjetos que permite instanciar un objeto de la clase ClsNotificacion y registrarlo en el n_s.

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 el Servidor de Notificaciones:
Para compilar el servidor de objetos:
```
javac -d ../bin servidorDeNotificaciones.servidor/*.java
```

Para el Servidor de Alertas:
Para compilar el servidor de objetos:
```
javac -d ../bin servidorDeAlertas.servidor/*.java
```

Para compilar el cliente de objetos:
```
javac -d ../bin cliente/*.java
```

3.c. Iniciar el registro.

Ubicarse en la carpeta bin/
Ejecutar el rmiregistry:
rmiregistry #_puerto

4. Iniciar la aplicación.

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

a. Ejecutar los servidores:
Comando general:
java Nombre_clase_servidor_obj

Tener en cuenta que el número de puerto del NS esta dado por el valor ingresado en ***#_puerto***

Ejecutar el ServidorNofiticaciones:
java servidorDeNotificaciones.servidor.ServidorDeObjetos

Ejecutar el ServidorAlertas:
java servidorDeAlertas.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, servidorDeNotificaciones y servidorDeAlertas 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.