Práctica 3: CallBack en Java RMI.

Diseñar un programa que permita enviar los datos básicos de un paciente y el indicador de su frecuencia cardiaca a un servidor de alertas, el cual analizara si la frecuencia cardiaca se encuentra fuera de un rango normal. Cada vez que se genere una alerta, el servidor de alertas debe enviar un mensaje de alarma a la habitación del paciente, para informar a los acompañantes el cambio de estado en los signos vitales del paciente. El mensaje a mostrar en la habitación del paciente es el siguiente:

> El paciente que se encuentra en la habitación [Número] presenta una frecuencia cardiaca de [Indicador de la frecuencia] la cual está fuera del rango normal

El sistema debe gestionar la información utilizando el modelo de RMI e implementándolo en lenguaje java. El rango normal de la frecuencia cardiaca por edad se muestra a continuación:

FRECUENCIA CARDIACA		
Grupo	**Edad**	**Latidos por minuto**
Adolescente	**13 – 16 años**	**70-80**
Adulto	**16 años y más**	**60-80**

En el cliente se debe crear un menú, tal como muestra la Figura 1:

========== Menú ==========

- *Enviar datos del paciente*
- *Salir*

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

Digite opción:

Figura 1: Contenido del Menú

La primera opción permite solicitar el no de habitación, edad, y frecuencia cardiaca.
Para observar el funcionamiento del programa debe ejecutar el ServidorDeObjetos y varios ClientesDeObjetos.

Desarrollo de la aplicación

La aplicación debe implementarse usando Java RMI, en un sistema operativo Linux o Windows. En la siguiente figura se observa un diagrama de contexto que muestra las operaciones que puede realizar el cliente. Las operaciones debe ser implementadas en un servidor, cada vez que se realice una petición en

el cliente y se reciba una petición en el servidor**, se debe crear un eco por medio de un mensaje en pantalla** en el cual se describe cual método se está invocando o ejecutando

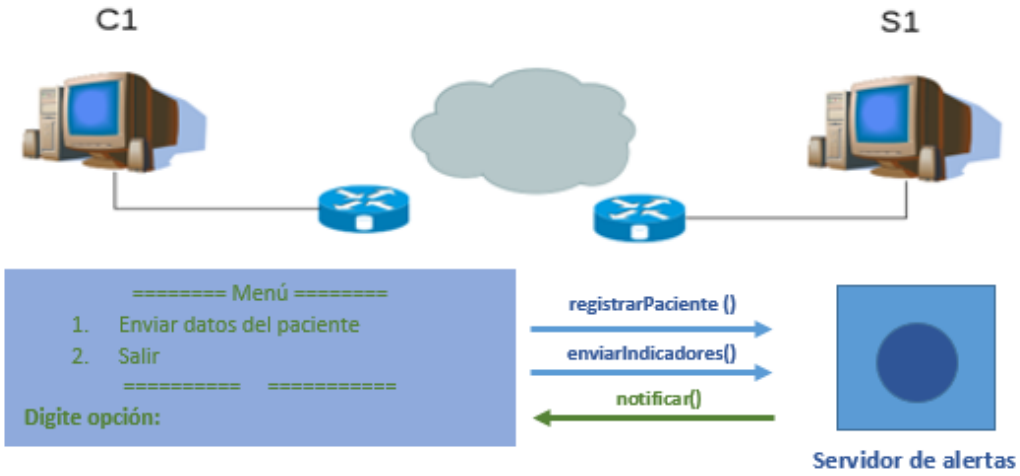

Figura 2: Diagrama de contexto de la aplicación

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato zip o rar y ser subida vía el enlace fijado en univirtual. El nombre del archivo comprimido debe seguir el siguiente formato lsd_rmi_p3_apellidoN1_apellidoN2.zip. Donde apellidoN1 corresponde al primer apellido de uno de los integrantes y apellidoN2 corresponde al primer apellido del segundo integrante del grupo.

Antes de iniciar se debe crear la siguiente estructura de directorios:

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 est_dir.bat para crear la estructura mostrada en la Figura 3.

```
lsd_rmi_p3_ape1_ape2/
├── bin
└── src
    ├── cliente
    ├── servidor
    └── sop_rmi
```

Figura 3: Estructura de directorios

1. Diseñar e implementar los componentes de la interface

1.a Definiendo la interface remota: Editar el archivo ServidorDeAlertasInt.java

```
package sop_rmi;

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

public interface ServidorDeAlertasInt extends Remote
{
        public boolean registrarPaciente(PacienteCllbckInt  objPaciente) throws RemoteException;
        public void enviarIndicadores(int noHabitacion,int edad, float frecuenciaCardiaca)throws RemoteException;

}
```

Figura 4. Interface ServidorDeAlertasInt

1.b Definiendo la clase que implementa la interface remota: Editar el archivo ServidorDeAlertasImpl.java

```
public class ServidorDeAlertasImpl extends UnicastRemoteObject implements ServidorDeAlertasInt {

    private List<PacienteCllbckInt> referenciasPacientes;

    public ServidorDeAlertasImpl() throws RemoteException
    {
        super();
        referenciasPacientes= new ArrayList();
    }

    @Override
    public void enviarIndicadores(int noHabitacion, int edad, float frecuenciaCardiaca)throws RemoteException
    {
        for(PacienteCllbckInt objUsuario: referenciasPacientes)
        {
            objUsuario.notificar("alerta generada en la habitacion: " + noHabitacion);
        }
    }

    @Override
    public synchronized boolean  registrarPaciente(PacienteCllbckInt objPaciente) throws RemoteException {
        System.out.println("Invocando al método registrar usuario desde el servidor");
        return referenciasPacientes.add(objPaciente);
    }
}
```

Figura 5. Clase ServidorDeAlertasImpl

1.c Definiendo la interface remota: Editar el archivo PacienteCllbckInt.java

```
package sop_rmi;

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

public interface PacienteCllbckInt extends Remote
{
    public void notificar(String mensaje) throws RemoteException;
}
```

Figura 6. Interface PacienteCllbckInt

1.d Definiendo la clase que implementa la interface remota: Editar el archivo PacienteCllbckImpl.java

```
package sop_rmi;

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

public class PacienteCllbckImpl extends UnicastRemoteObject implements PacienteCllbckInt
{
    public PacienteCllbckImpl() throws RemoteException
    {
        super();
    }

    @Override
    public void notificar(String mensaje) throws RemoteException {
        System.out.println("Mensaje enviado del servidor: " + mensaje);
    }

}
```

Figura 7. Clase PacienteCllbckImpl

2. Crear 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. El nombre del objeto remoto debe ser **ObjetoRemotoGestionPacientes**. La figura 8 describe un ejemplo del código fuente del ClienteDeObjetos.

```
public class ClienteDeObjetos
{
    public static void main(String[] args)
    {
        try
        {
            ServidorDeAlertasInt servidor;
            int edad, frecuenciaCardiaca, noHabitacion, 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();

            servidor = (ServidorDeAlertasInt) UtilidadesRegistroC.obtenerObjRemoto(numPuertoRMI

            PacienteCllbckImpl nuevoUsuario= new PacienteCllbckImpl();
            servidor.registrarPaciente(nuevoUsuario);

            System.out.println("No de habitación: ");
            noHabitacion=UtilidadesConsola.leerEntero();
            System.out.println("Digite la edad: ");
            edad=UtilidadesConsola.leerEntero();
            System.out.println("Digite la frecuencia cardica: ");
            frecuenciaCardiaca=UtilidadesConsola.leerEntero();
            servidor.enviarIndicadores(noHabitacion,edad, frecuenciaCardiaca);
```

Creación del objeto remoto del lado del cliente y envio de su referencia remota al servidor.

Figura 8. Cliente de objetos

3. Crear el código del servidor de objetos

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. El nombre del objeto remoto debe ser **ObjetoRemotoGestionPacientes**. La figura 9 describe un ejemplo del código fuente del ServidorDeObjetos.

```
public class ServidorDeObjetos {
    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 = UtilidadesConsola.leerCadena();
        System.out.println("Cual es el número de puerto por el cual escucha el rmiregistry ");
        numPuertoRMIRegistry = UtilidadesConsola.leerEntero();
        try
        {
            ServidorDeAlertasImpl objRemoto = new ServidorDeAlertasImpl();
            UtilidadesRMIServidor.ArrancarNS(numPuertoRMIRegistry);
            UtilidadesRMIServidor.RegistrarObjetoRemoto(objRemoto, direccionIpRMIRegistry, numPuertoRMIRegistry,"ObjetoRemotoGestionPacientes");
            System.out.println("Objeto remoto registrado, esperado peticiones ...");
        } catch (Exception e)
        {
            System.err.println("No se pudo Arrancar el NS o Registrar el objeto remoto");
        }
    }
}
```

Figura 9. Servidor de objetos

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

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

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

5. 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.