Práctica 3: CallBack en Java RMI.

La División de Gestión de la Recreación y el Deporte de la Universidad del Cauca es una dependencia adscrita a la Vicerrectoría de Cultura y Bienestar, definida como el componente operativo del Sistema Institucional de Deporte y Recreación Universitarios. Esta dependencia cuenta con 3 programas: Formación deportiva, Estilos de vida saludable y Unicauca activa de corazón. En este caso se va a centrar en el programa de Estilos de vida saludable. El programa Estilos de vida saludable se subdivide en 3 sub-programas: Programa Hora saludable, Acondicionamiento físico para estudiantes, Gimnasio para los administrativos.
Como soporte del programa de Hora saludable se cuenta con el siguiente personal: una secretaria y profesional en acondicionamiento físico. El registro de usuarios (docentes y administrativos) está a cargo de la secretaria, el profesional en acondicionamiento físico encargado de realizar valoraciones físicas elaboración de los programas físicos de los usuarios y registro de asistencia.
La División de Gestión de la Recreación y el Deporte de la Universidad del Cauca requiere que los usuarios tales como: secretaria y profesional de acondicionamiento físico puedan ser registrados en el servidor de gestión de personal y una vez registrados que este personal pueda abrir sesión en la aplicación.

Para el registro y consulta del personal los datos a registrar corresponden a: tipo de identificación (cc, ti, pp), número de identificación, nombre completo, ocupación (secretaria, paf), usuario, clave. El sistema debe gestionar la información utilizando el modelo de RMI e implementándolo por medio de Java RMI.
Al iniciar el programa se debe desplegar el siguiente menú:

```
 ==== Menú Inicio ====
* 1-Abrir sesión      *
* 2-Salir             *
 ======================
```

Figura 1. Menú de ingreso

La primera vez que se ejecuta la aplicación se debe abrir sesión para el administrador, ingresando las credenciales del administrador del programa (usuario, clave) y el número de identificación. El Menú del administrador del programa debe contar con 3 opciones: registrar personal, consultar personal y salir. Estas opciones serán controladas mediante un Menú, tal como muestra la Figura 2:

```
====== Menú Admin ======
* 1-Registrar personal *
* 2-Consultar personal *
* 3-Salir              *
 ======================
```

Figura 2. Menú del administrador

La primera opción permite registrar todos los datos de un usuario. En el lado del servidor los datos de los usuarios serán almacenados en un arreglo. La máxima cantidad de usuarios a registrar será de 5 usuarios.
Si el registro de usuario es exitoso se debe emitir un mensaje: ** Usuario registrado exitosamente **.
Si el registro el registro no es exitoso se debe emitir el mensaje: ** Usuario NO registrado, se alcanzó la cantidad máxima de usuarios a registrar **.

En la segunda opción si la consulta es exitosa se debe desplegar la siguiente información del usuario: Tipo de identificación, Identificación, Nombre completo, Role, Usuario.
Si la consulta no es exitosa se debe desplegar el mensaje: **Usuario NO encontrado**.

Para abrir sesión de un personal con ocupación secretaria o paf se deben ingresar las credenciales del personal (usuario, clave) y el número de identificación. Si el usuario que ingresa es una secretaria o el paf y ya están registrados se debe desplegar el menú que se muestra en la Figura 1.
Si las credenciales son correctas y la ocupación es una secretaria se debe desplegar el siguiente menú (Figura 3):

```
=== Menú Secretaria ===
* 1-Registrar usuario *
* 2-Consultar usuario *
* 3-Salir             *
======================
```

Figura 3. Menú de la secretaria

Si las credenciales son correctas y la ocupación es un paf se debe desplegar el siguiente menú (Figura 4):

```
====== Menú PAF =========
* 1-Valorar PAF          *
* 2-Registrar asistencia *
* 3-Salir                *
=========================
```

Figura 4. Menú del PAF

Si las credenciales no son correctas, en la pantalla del nodo cliente se le mostrará el siguiente mensaje:

** El personal [usuario] no está autorizado para ingresar al sistema. **
** Verificar que el usuario y la clave sean las correctas. **

Mensaje Callback:
Para el caso de un ingreso exitoso, paralelamente en la ventana del administrador se debe desplegar el siguiente mensaje:
*** El usuario [nombre completo] Identificado con id [id] Ingresó a la aplicación. ***

Para observar el funcionamiento del programa debe ejecutar el nodo ServidorDeObjetos1 y 2 nodos ClientesDeObjetos, como mínimo.

Desarrollo de la aplicación

La aplicación debe implementarse usando Java RMI, en un sistema operativo GNU/Linux o Windows. En la figura 5 se observa un diagrama de contexto que muestra las operaciones que puede realizar el cliente. Las operaciones deben 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

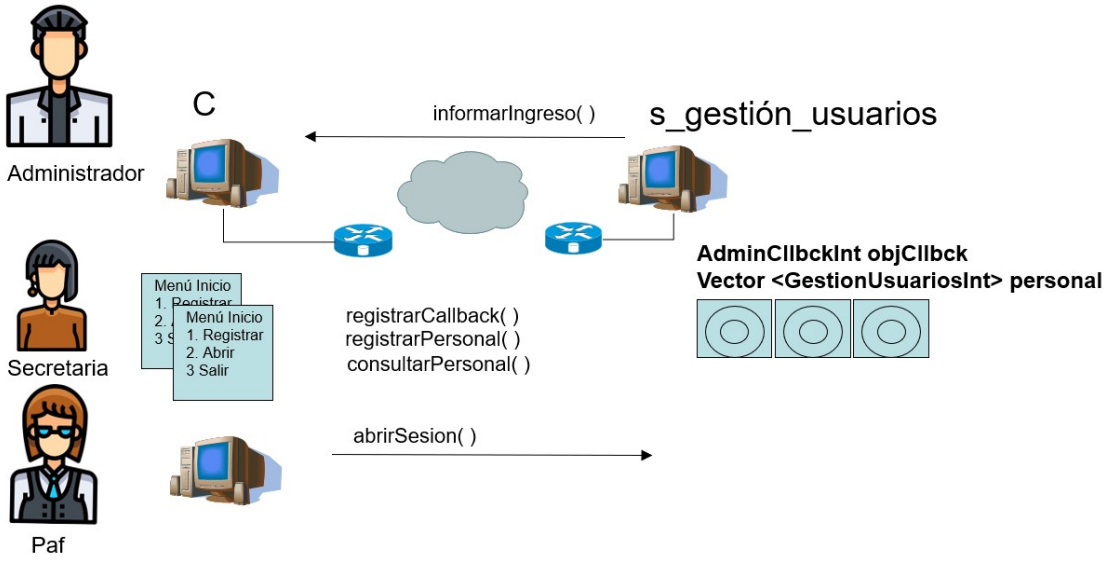

Figura 5: Diagrama de contexto de la aplicación

Tener en cuenta: La solución de este ejercicio debe ser comprimida en formato rar y ser subida vía el enlace fijado en univirtual. El nombre del archivo comprimido debe seguir el siguiente formato lsdA_rmi03_gx. Donde gx corresponde al nombre del grupo asignado en este curso.

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

Antes de iniciar, estructurar una carpeta de trabajo donde se ubicarán 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. Ver Figura 6.

```
lsdA-rmi03-gx/
├── bin
└── src
    ├── cliente
    │   ├── sop_rmi
    │   └── utilidades
    └── s_gestion_riesgos
        ├── dto
        ├── sop_rmi
        └── utilidades
```

Figura 6: Estructura de directorios

1. Diseñar e implementar los componentes de la interface

1.a Definiendo la interface remota: Editar el archivo GestiónUsuariosInt.java e incluir el contenido que se muestra en la figura 7.

```
/* GestionUsuariosInt.java*/

package s_gestion_usuarios.sop_rmi;


import java.rmi.Remote;
import java.rmi.RemoteException;
import s_gestion_usuarios.dto.PersonalDTO;
import s_gestion_usuarios.dto.CredencialDTO;
import cliente.sop_rmi.AdminCllbckInt;

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

    public boolean registrarUsuario(PersonalDTO objPersonal) throws RemoteException;
    public PersonalDTO consultarUsuario(int id) throws RemoteException;
    public void registrarCallback(AdminCllbckInt objAdmin) throws RemoteException;
    public boolean abrirSesion(CredencialDTO objCredencial) throws RemoteException;
}
```

Figura 7. Interface remota GestionUsuariosInt

1.b Implementando las clases DTO: Editar el archivo *PersonalDTO.java* y *CredencialDTO.java*. Se utilizará el patrón de diseño DTO (Data Transfer Object) que permite tener objetos por medio de los cuales se transporta datos entre procesos. Por tanto, se debe implementar una clase *PersonalDTO* que contendrá los atributos de un usuario. La estructura de esta clase se muestra en la siguiente Figura 8, y se debe implementar otra clase *CredencialDTO* que contendrá los atributos **usuario**, **clave**. La estructura de esta clase se muestra en la siguiente Figura 9.

```
package s_gestion_usuarios.dto;

import java.io.Serializable;

public class PersonalDTO implements Serializable
{
    private String tipo_id;//cc,ti,pp
    private int id;
    private String nombreCompleto;
    private String ocupacion;//secretaria, paf
    private String usuario;
    private String clave;

    public PersonalDTO(String tipo_id,int id,String nombreCompleto,String ocupacion,String usuario,String clave) {
      this.tipo_id = tipo_id;
      this.id = id;
      this.nombreCompleto = nombreCompleto;
      this.ocupacion = ocupacion;
      this.usuario = usuario;
      this.clave = clave;
    }

    public String getTipo_id() {···
    public void setTipo_id(String tipo_id) {···
    public int getId() {···
    public void setId(int id) {···
    public String getNombreCompleto() {···
    public void setNombreCompleto(String getNombreCompleto) {···
    public String getOcupacion() {···
    public void setOcupacion(String ocupacion) {···
    public String getUsuario() {···
    public void setUsuario(String usuario) {···
    public String getClave() {···
    public void setClave(String clave) {···
}
```

Figura 8. Clase PersonalDTO

```java
package s_gestion_usuarios.dto;
import java.io.Serializable;

public class CredencialDTO implements Serializable
{
    private String usuario;
    private String clave;

    public CredencialDTO(String usuario,String clave) {
      this.usuario = usuario;
      this.clave = clave;
    }

    public String getUsuario() {
        return this.usuario;
    }
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
    public String getClave() {
        return clave;
    }
    public void setClave(String clave) {
        this.clave = clave;
    }
}
```

Figura 9. Clase CredencialDTO

1.c Definiendo la clase que implementa la interface remota: Editar el archivo GestionUsuariosImpl.java e implementar los métodos de la interface remota GestionUsuariosInt (ver Figura 10).

```java
package s_gestion_usuarios.sop_rmi;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import cliente.sop_rmi.AdminCllbckInt;
import s_gestion_usuarios.dto.PersonalDTO;
import s_gestion_usuarios.dto.CredencialDTO;
import s_gestion_usuarios.utilidades.UtilidadesRegistroC;


public class GestionUsuariosImpl extends UnicastRemoteObject implements GestionUsuariosInt
{
    private ArrayList<PersonalDTO> personal;
    private int contador=0;
    private int siAdmin=0;
    private AdminCllbckInt objCllbck;

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

    @Override
    public boolean registrarUsuario(PersonalDTO objPersonal) throws RemoteException {···

    @Override
    public PersonalDTO consultarUsuario(int id) throws RemoteException {···

    @Override
    public void registrarCallback(AdminCllbckInt objCllbck) throws RemoteException{···

    @Override
    public boolean abrirSesion(CredencialDTO objCredencial) throws RemoteException {···

    public int buscarUsuario(int id){···

    public int buscarCredencial(CredencialDTO objCredencial){···

}
```

Figura 10. Clase GestionUsuariosImpl

1.d Definiendo la interface remota: Editar el archivo AdminCllbckInt.java e incluir el contenido que se muestra en la Figura 11, el cual corresponde al objeto callback.

```
package cliente.sop_rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;
import s_gestion_usuarios.dto.IngresoDTO;

public interface AdminCllbckInt extends Remote{
        public void informarIngreso(String nombreC, int id) throws RemoteException;
}
```

Figura 11. Interface remota AdminCllbckInt

1.e Definiendo la clase que implementa la interface remota: Editar el archivo AdminCllbckImpl.java que implementa l interface remota AdminCllbckInt

```
package cliente.sop_rmi;

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

public class AdminCllbckImpl extends UnicastRemoteObject implements AdminCllbckInt
{

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

    @Override
    public void informarIngreso(String nombreCompleto, int id) throws RemoteException {…

}
```

Figura 12. Clase AdminCllbckImpl

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 **ObjetoRemotoPersonal**. La Figura 13 describe un bloque del código fuente del ClienteDeObjetos, donde se instancia el objeto callback y se envía su referencia remota para que sea registrado en el servidor de gestión de riesgos.

```
System.out.println("*** Registrando objeto Callback... ");
AdminCllbckImpl objAdmin=new AdminCllbckImpl();
objRemoto01.registrarCallback(objAdmin);
```

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

Figura 13. Cliente de objet

3. Crear el código del servidor de objetos

Editar un archivo denominado ServidorDeObjetos01.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 **ObjetoRemotoPersonal**.

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 en el servidor:

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

Para compilar el sop_rmi en el cliente:

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

Para compilar el servidor de objetos:

```
javac -d ../bin s_gestion_usuarios/*.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 s_gestion_usuarios.ServidorDeObjetos01
```

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 s_gestion_usuarios respectivamente. Los archivos fuentes no deben estar vinculados con ningún IDE de desarrollo.