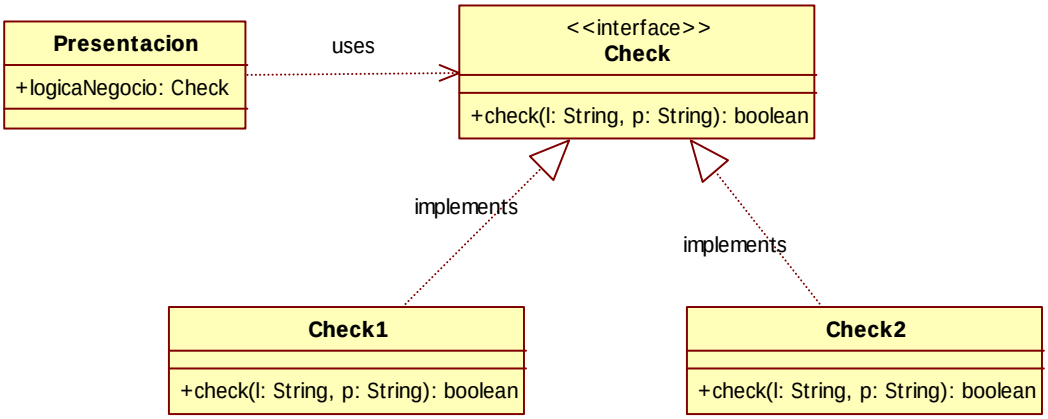
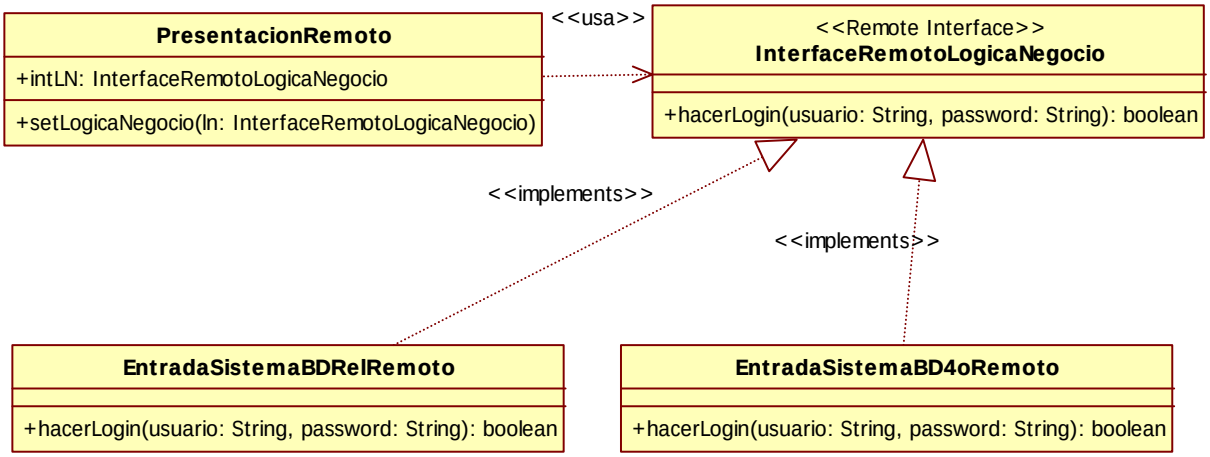
Introducción

En este laboratorio desplegaremos en 3 niveles físicos una aplicación que verifica si una cuenta y un password son correctos, basada en la que fue presentada en el laboratorio "Separación entre Presentación y Lógica del Negocio".



El acceso al servidor de la lógica del negocio desde la presentación (cliente) se realizará utilizando RMI. Este es el motivo por el que decimos que esta aplicación "se basará" en las anteriores, ya que el uso de RMI requiere que la interfaz "Check" anteriormente utilizada no sirva, ya que debe ser una interfaz remota.

Para facilitar la tarea de programación de todo aquello que tiene que ver con RMI, en este laboratorio se proporciona una solución de una aplicación que verifica si una cuenta y un password son correctos, comprobándolo en una base de datos relacional. ***Lo que se pide es, realizar una aplicación similar que verifique si la cuenta y el password son correctos, pero comprobándolo en una base de datos orientada a objetos bd4o.*** La implementación de esa lógica del negocio concreta se basará, por tanto, en la desarrollada en el punto 6 del laboratorio "Implementación del nivel de datos usando db4o". Se proporciona el código de las clases: PresentacionRemoto, InterfaceRemotoLogicaNegocio y EntradaSistemaBDRelRemoto, y se os solicita que implementéis el código de EntradaSistemaBD4oRemoto.



Objetivos

Los objetivos del laboratorio son los siguientes:

- Saber construir y desplegar una aplicación en 3 niveles físicos utilizando RMI y BD4o.

Actividades a realizar

Os proponemos las siguientes 4 actividades.

1. Instalar y ejecutar la aplicación de entrada al sistema usando una BDR

Los pasos a realizar serán los siguientes:

a) Descomprimir el contenido del fichero LabRMI.zip disponible en el Moodle de la asignatura. Para tener menos problemas con los nombres de los ficheros incluidos en el código, se recomienda que se haga en el directorio C:\

b) Desde Eclipse, importar
File => Import => General => Existing Projects into Workspace => Next => Select root directory => Browse => Seleccionar directorio C:\LabRMI y proyecto LabRMI => Finish

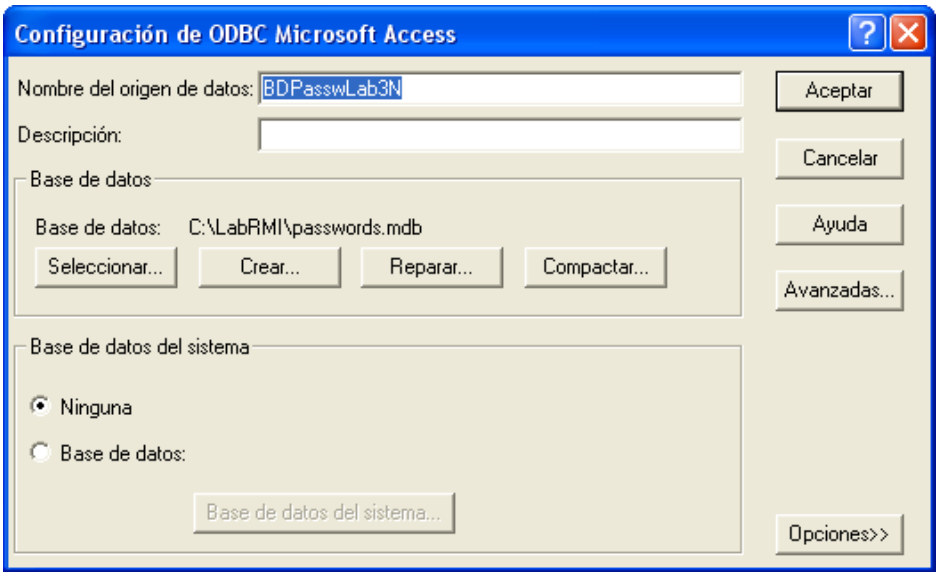
c) Asegurarse de que el fichero con la política de seguridad que se utiliza tanto en la clase servidora RMI (EntradaSistemaBDRelRemoto) como en la cliente (PresentacionRemoto) es el fichero java.policy que se ha extraído de LabRMI.zip. En nuestro caso, se supone que se habrá dejado en `C:\LabRMI\java.policy`

```
System.setProperty("java.security.policy","C:\\LabRMI\\java.policy");
```

d) Definir una fuente de datos ODBC que se llame `BDPasswLab3N` y que esté asociada al fichero con la BD Access, passwords.mdb, también extraído de LabRMI. Esta fuente de datos es la utilizada en la clase servidora RMI (EntradaSistemaBDRelRemoto)

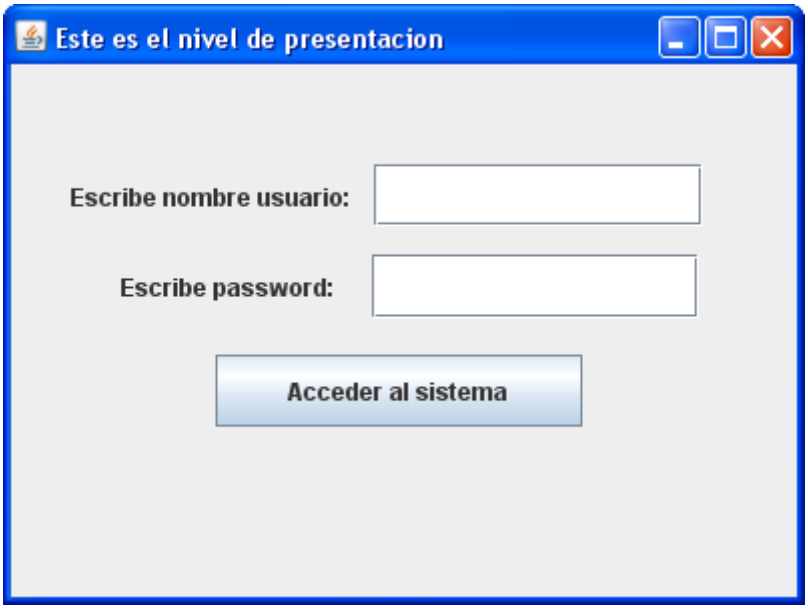
```
o=DriverManager.getConnection("jdbc:odbc:BDPasswLab3N");
```

Inicio => Panel de Control => Herramientas Administrativas => Orígenes de datos ODBC => Agregar => Microsoft Access Driver (*.mdb) => Finalizar => Escribir “BDPasswLab3N” en “Nombre del origen de datos” => Seleccionar => seleccionar el fichero C:\LabRMI\passwords.mdb => Aceptar



e) Ejecutar la clase remota EntradaSistemaBD y comprobar que muestra los mensajes de que ha cargado el driver y lanzado el objeto servidor remoto.

f) Ejecutar la clase cliente PresentacionRemoto y probar que funciona correctamente, de acuerdo a los valores existentes en la base de datos passwords.mdb



Para que deje acceder al sistema, debe coincidir el nombre del usuario y el password y no haber realizado 3 intentos fallidos de entrada al sistema. Cada vez que deja entrar al sistema, se pone a 0 el número de intentos fallidos, y cada vez que se intenta entrar al sistema con un usuario existente y un password incorrecto, se incrementa en 1 el número de intentos fallidos.

2. Examinar el código de la aplicación

En esta parte, simplemente os solicitamos que reviséis las partes de RMI:

a.- Definir la interfaz remota
b.- Implementar la interfaz remota (clase remota)
c.- En el servidor, registrar un objeto de la clase remota
d.- En el cliente, localizar y ejecutar el objeto remoto
e.- Crear el gestor de seguridad y definir la política de seguridad.
f.- Comprobar las clases usadas en métodos remotos son serializables

a) Definir la interfaz remota

Comprobar que se ha definido una interfaz remota, que extiende `java.rmi.Remote` y que declara todos sus métodos susceptibles de lanzar la excepción `java.rmi.RemoteException`

```
package LabRMI;
import java.rmi.*;

public interface InterfaceRemotoLogicaNegocio extends Remote
{
   boolean hacerLogin(String usuario,String password)
      throws RemoteException;
}
```

b) Implementación de la clase remota

Comprobar que la clase remota implementa la interfaz remota, que extiende de `java.rmi.server.UnicastRemoteObject` y que todos sus métodos (incluido el constructor de la clase) se declaran como que pueden lanzar `java.rmi.RemoteException`

```
public class EntradaSistemaBDRelRemoto
      extends UnicastRemoteObject
      implements InterfaceRemotoLogicaNegocio


public EntradaSistemaBDRelRemoto() throws RemoteException { ... }

public boolean hacerLogin(String a,String b) throws RemoteException
{ ... }
```

c) En el servidor, registrar un objeto de la clase remota

Comprobar que en el main de la clase remota se crea una instancia del objeto remoto (`new EntradaSistemaBDRelRemoto()`), se crea el registro de nombres en el puerto *numPuerto* del servidor (`java.rmi.registry.LocateRegistry.createRegistry(numPuerto)`), y se le dice al registro de nombres que exporte que el objeto se va a poder invocar de manera remota dándole un nombre de servicio (`Naming.rebind`).

```
EntradaSistemaBDRelRemoto objetoServidor;
try {

// ...
objetoServidor = new EntradaSistemaBDRelRemoto();

 try { java.rmi.registry.LocateRegistry.createRegistry(numPuerto);  }
catch (Exception e)
      {System.out.println(e.toString()+"\nSe supone que el error es
porque el rmiregistry ya estaba lanzado");}

      // Registrar el servicio remoto
Naming.rebind("//localhost:"+numPuerto+"/"+nombreServicio,
              objetoServidor);

} catch (Exception e)
{System.out.println("Error al lanzar el servidor: "+e.toString());}
}
```

d) En el cliente, localizar y ejecutar el objeto remoto

Comprobar que en el main de la clase cliente PresentacionRemoto se busca la referencia del objeto remoto (`Naming.lookup`), el cual se debe estar ejecutando en la máquina *maquinaRemota* y se ha debido de registrar en el registro de nombres lanzado en esa misma máquina en el puerto *numPuerto.* Por último, dicho objeto remoto se asigna como la lógica del negocio (`p.setLogicaNegocio`) del cliente.

```
try{
  p.setLogicaNegocio(
(InterfaceRemotoLogicaNegocio)
Naming.lookup("rmi://"+maquinaRemota+":"+numPuerto+"/"+nombreServicio)
);

} catch(Exception e) {System.out.println("Error al conseguir la lógica
del negocio: "+ e.toString()); }
```

e) Crear el gestor de seguridad y definir la política de seguridad

Tanto en el main del cliente como del servidor hay que crear un gestor de seguridad y definir una política de seguridad.

```
System.setProperty("java.security.policy",
                   "C:\\LabRMI\\java.policy");
System.setSecurityManager(new RMISecurityManager());
```

En nuestro caso, el fichero con la política de seguridad va a dar todos los permisos al gestor de seguridad (para que permita cargar clases en tiempo de ejecución enviadas desde el servidor al cliente, al crear y utilizar sockets entre el cliente y servidor, etc. todo ello necesario para que funcione una aplicación con RMI. Se podrían definir permisos de seguridad más específicos para todos estos casos.

El contenido del fichero java.policy es el siguiente:

```
grant {
permission java.security.AllPermission;
};
```

f) Comprobar que las clases usadas en métodos remotos son serializables

En el único método remoto utilizado (`public boolean hacerLogin(String a,String b)`), sólo se utiliza la clase String, que está declarada como Serializable (tal y como se puede comprobar en la API de Java).

Nótese, que si hubiéramos utilizado una clase nuestra, por ejemplo Vuelo, y quisiéramos utilizarla en un método remoto, deberíamos definir dicha clase como serializable. Simplemente habría que declararla así:

```
public class Vuelo implements java.io.Serializable {...}
```

3. Probar a ejecutar la aplicación en 3 máquinas diferentes (3 niveles físicos)

Para ejecutar la presentación y la lógica del negocio en máquinas diferentes habría que asegurarse de lo siguiente:

- Desde el cliente PresentacionRemoto se invoca a la lógica del negocio ejecutada en otra máquina. Se puede hacer asignando a los atributos *`maquinaRemota`* y *`numPuerto`* los números de IP y puerto de otra máquina donde hayamos lanzado previamente el servidor remoto `EntradaSistemaBDRelRemoto`.

Para que la lógica del negocio y el nivel de datos (la BD relacional) se encuentren en distintas máquinas habría que:

- En este caso, redefinir la fuente de datos ODBC `BDPasswLab3N` y seleccionar el fichero passwords.mdb que se encuentra en otro ordenador. Para ello, en el ordenador remoto con la BD hay que

compartir la carpeta donde se encuentra el fichero con la BD (dando permisos de lectura y escritura), y en el ordenador con la lógica del negocio habría que buscar y asignar dicha carpeta compartida en la red y asignarla a una unidad. Aunque claro está, también hay más posibilidades como dejar en dropbox la BD si tenemos problemas con compartir carpetas y darles permisos apropiados.

4. Implementación de la clase EntradaSistemaBD4oRemoto

Se pide implementar una nueva lógica del negocio, accesible igualmente mediante RMI, y que utilice para la persistencia una base de datos BD4o.

Dicha lógica del negocio debe funcionar de igual manera que EntradaSistemaBDRelRemoto, esto es, debe comprobar además que no se ha llegado al máximo de intentos fallidos (3), incrementando dicho contador cada vez que se produzca un nuevo intento fallido o poniendo a 0 el contador cada vez que se entre al sistema de manera satisfactoria.

Ya que hay dos partes diferenciadas: parte cliente y parte servidora, en principio no tendrían que estar en un mismo proyecto (del que podría generarse un único fichero JAR). Sería mucho mejor que se definieran 2 proyectos diferentes, uno para ejecutar en el servidor y otro en el cliente. Y no olvidar, que al menos en ambos sí debe estar la interfaz remota.