

Documentación:

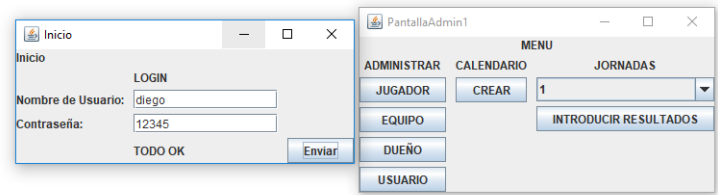
Aplicación

RETO 1- GRUPO 1 –DIEGO GARCIA –IZARO
GARCIA

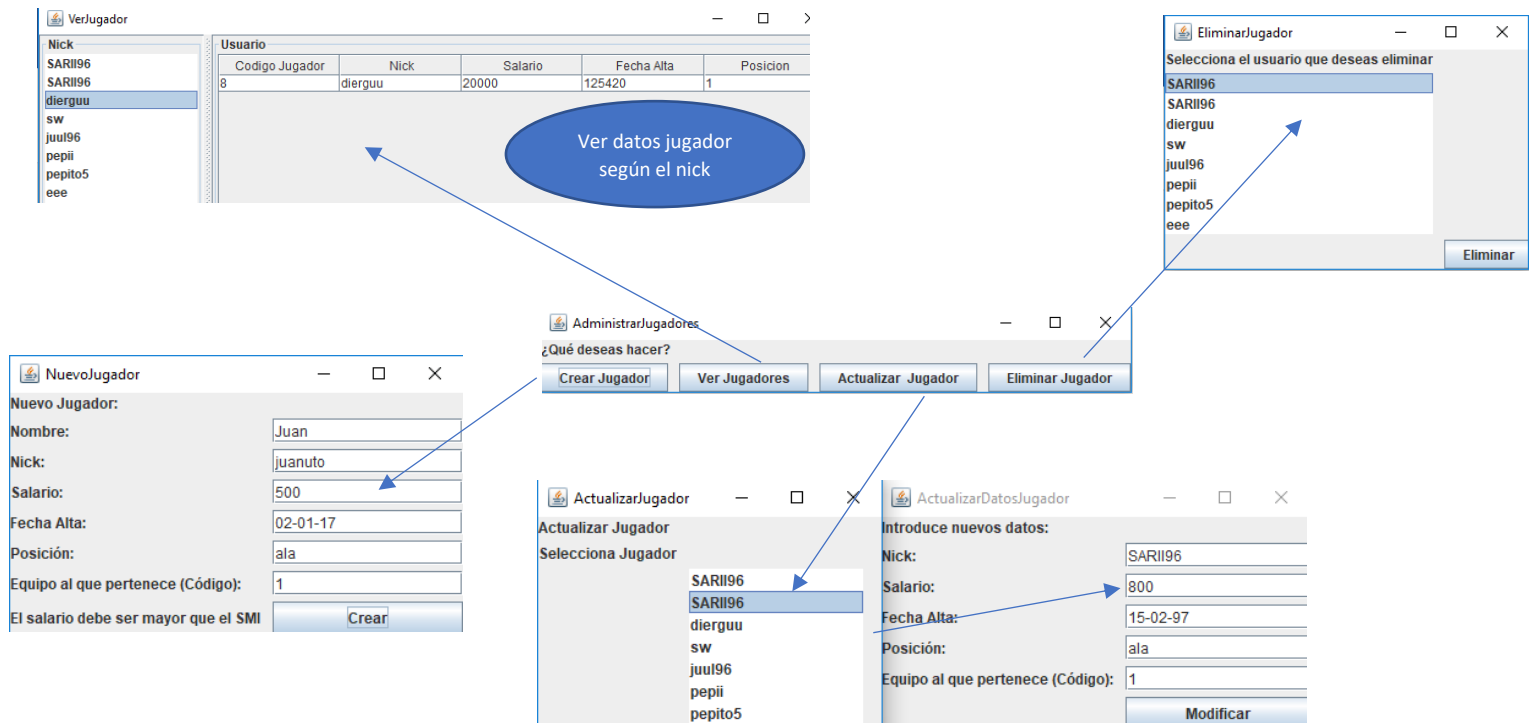
- **Índice:**
 1. **Esquema visual de uso**
 2. **Dueño**
 3. **Pruebas con JUnit**

1. Esquema visual de Uso:

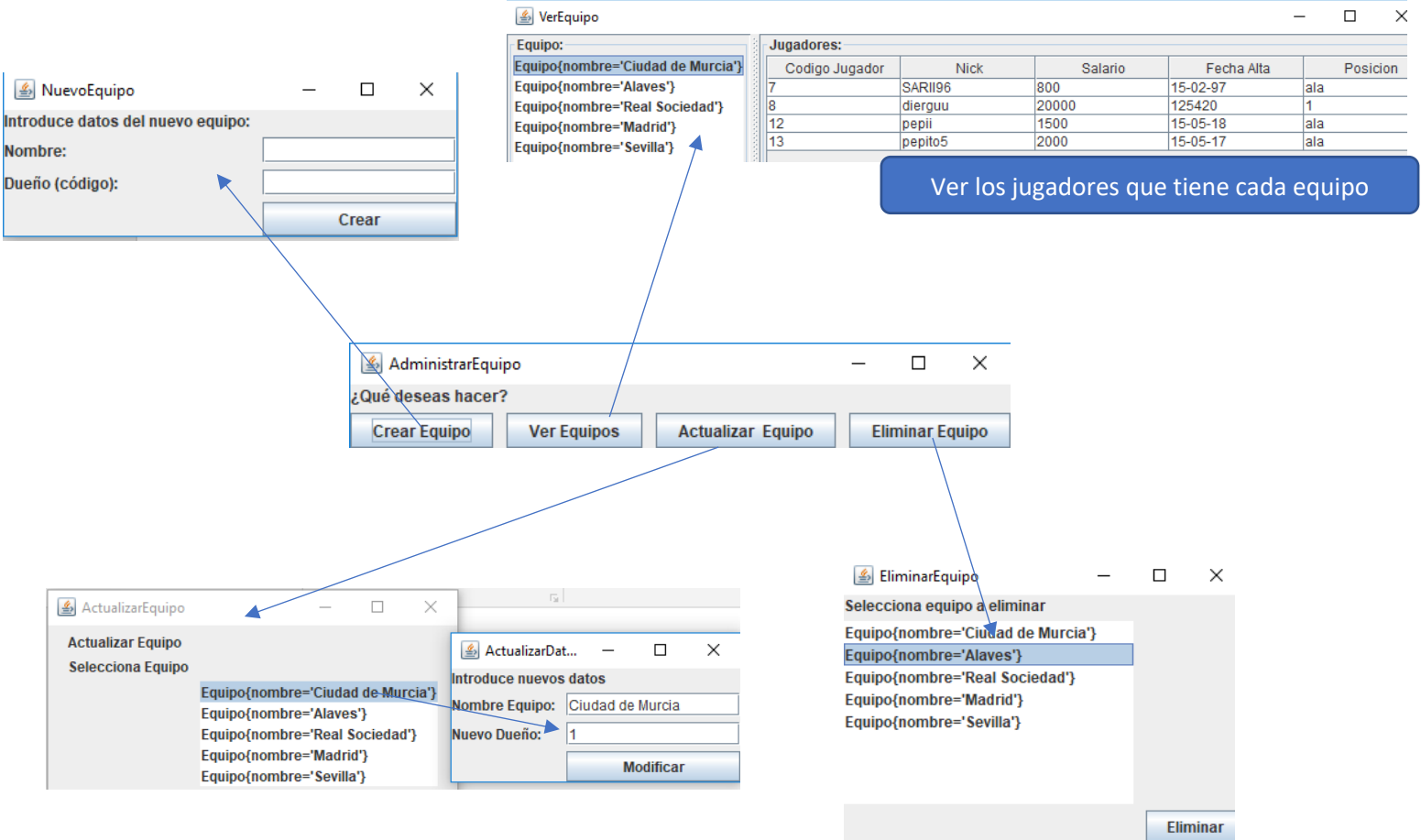
Login y pantalla administrador



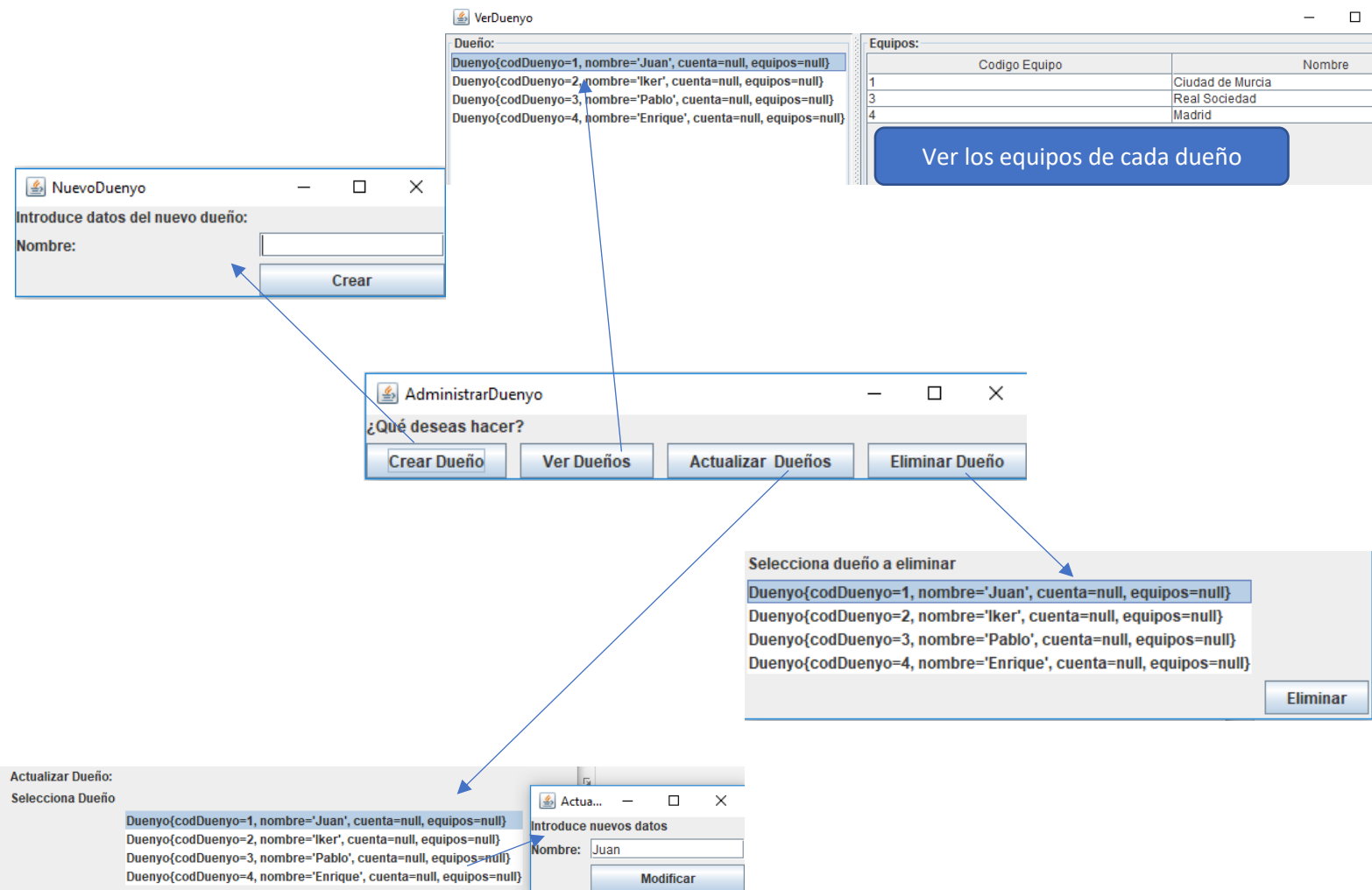
Administrar Jugador



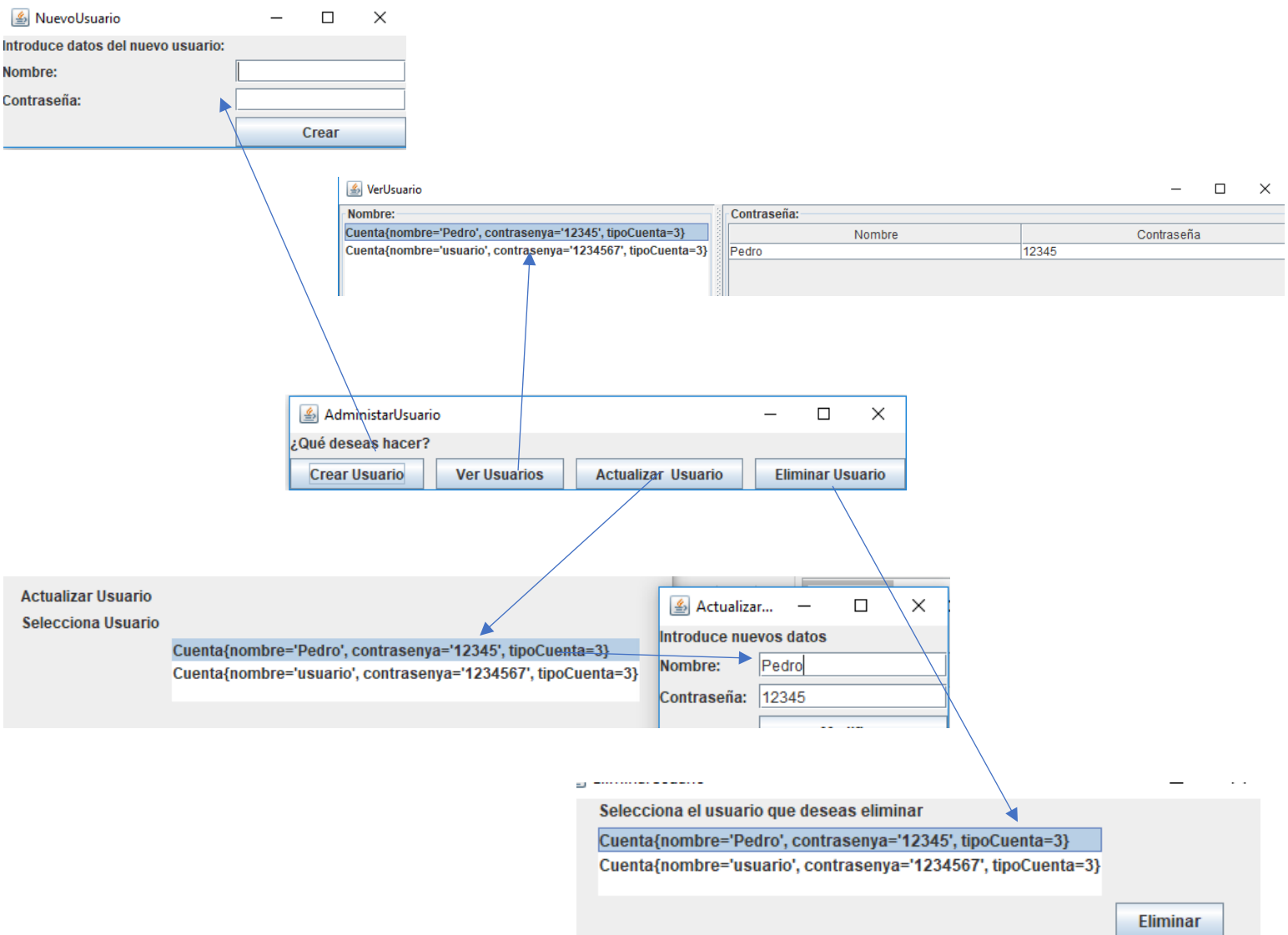
Administrar Equipo



Administrar Dueño



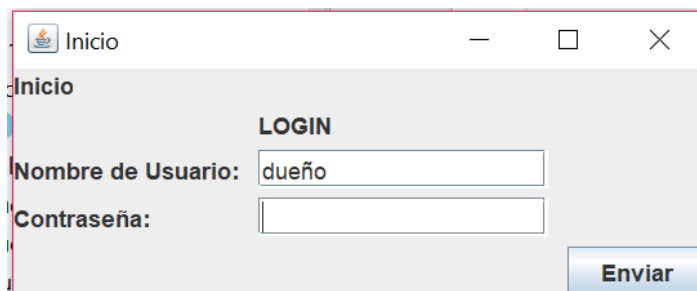
Administrar Usuario



2. Dueño

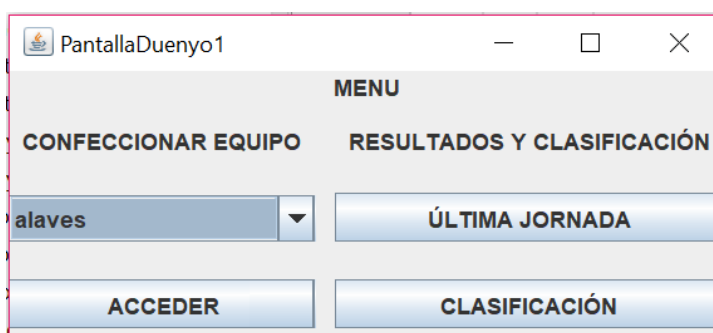
Un dueño va a tener la posibilidad de comprar o vender jugadores para confeccionar sus equipos teniendo en cuenta unas restricciones.

Como todos los perfiles accederá desde la pantalla login con su usuario y contraseña.



The screenshot shows a window titled 'Inicio' with a 'LOGIN' section. It contains two input fields: 'Nombre de Usuario:' with the text 'dueño' and 'Contraseña:' which is empty. A blue 'Enviar' button is located at the bottom right.

Después de acceder al programa, aparecerá una pantalla nueva con varias opciones.



The screenshot shows a window titled 'PantallaDuenyo1' with a 'MENU' section. It has two main options: 'CONFECCIONAR EQUIPO' and 'RESULTADOS Y CLASIFICACIÓN'. Under 'CONFECCIONAR EQUIPO', there is a dropdown menu showing 'alaves' and a blue 'ACCEDER' button. Under 'RESULTADOS Y CLASIFICACIÓN', there is a blue 'ÚLTIMA JORNADA' button and a blue 'CLASIFICACIÓN' button.

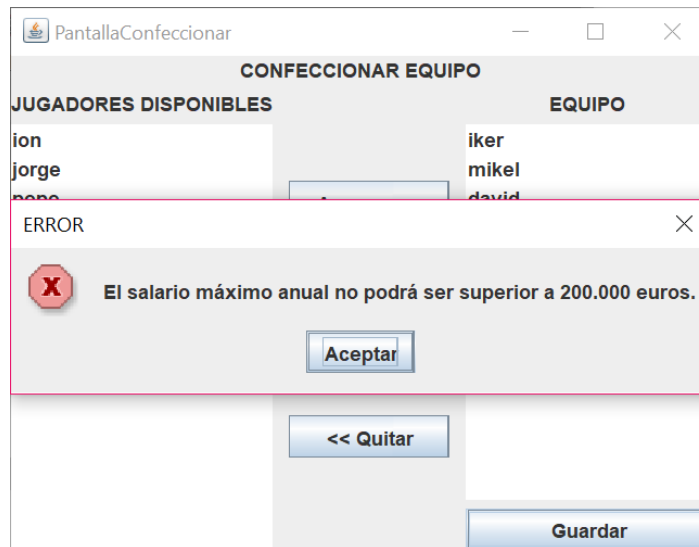
Aquí, el dueño podrá ver sus equipos, la última jornada y la clasificación. Por ejemplo, tiene el equipo Alaves. Si hace click en el botón Acceder podrá ver por un lado los jugadores del Alaves y en la otra lista verá los jugadores que estén disponibles o en otros equipos. De este modo, el dueño podrá comprar y vender los jugadores que el quiera. Siempre y cuando tenga en cuenta las restricciones.



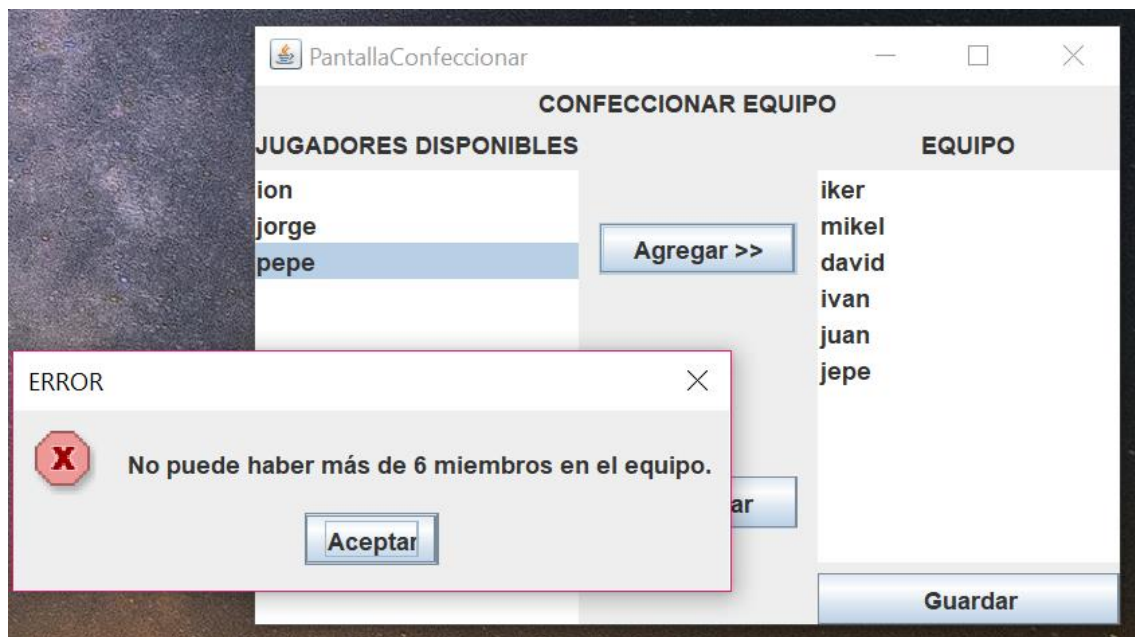
The screenshot shows a window titled 'PantallaConfeccionar' with a 'CONFECCIONAR EQUIPO' section. It is divided into two columns: 'JUGADORES DISPONIBLES' on the left and 'EQUIPO' on the right. The 'JUGADORES DISPONIBLES' column lists 'ion', 'jorge', 'pepe', and 'jepe'. The 'EQUIPO' column lists 'iker', 'mikel', 'david', 'ivan', and 'juan'. Between the columns are two buttons: 'Agregar >>' and '<< Quitar'. A blue 'Guardar' button is at the bottom right.

Las restricciones de las que os hablaba son estas:

- El salario anual no puede ser superior a 200.000 euros.



- Los equipos están formados como máximo por seis miembros.



3. Pruebas con JUnit

- a. Se ha utilizado JUnit 4 para comprobar que los métodos más complejos o específicos funcionan correctamente. Las pruebas se encuentran en la carpeta *test*. Hemos realizado pruebas en los siguientes métodos:

i. **salarioTotal()** en JugadorBD:

1. Este método calcula el salario total de un equipo determinado, pasándole el código del equipo como parámetro.

```
//SALARIO TOTAL DE UN EQUIPO
public static int salarioTotal(int codEquipo) {

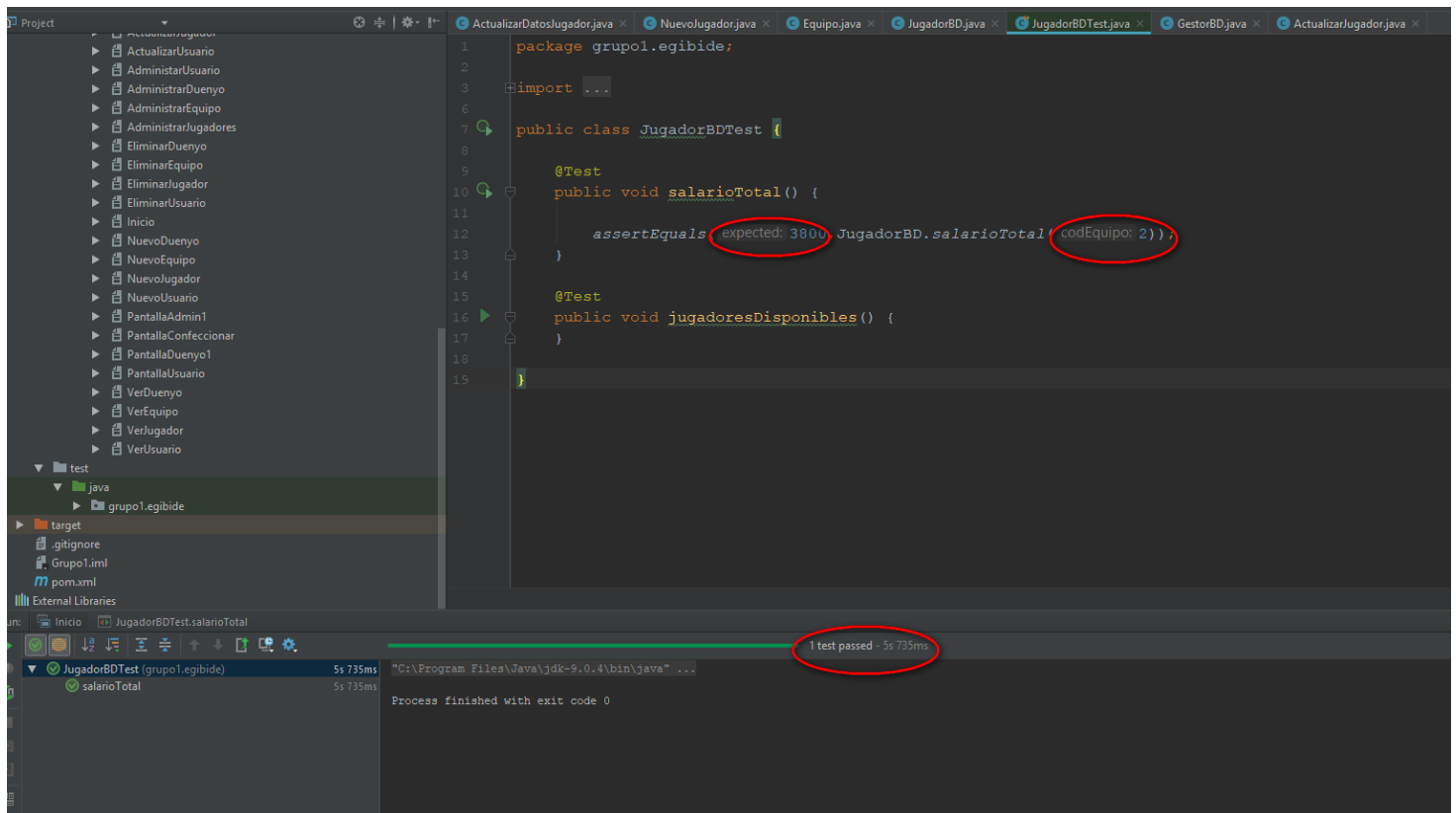
    Connection conexion = GestorBD.conectar();
    int sum = 0;
    try {
        Statement st = conexion.createStatement();
        ResultSet res = st.executeQuery(sql: "SELECT SUM(salario) FROM Jugador WHERE Equipo_codEquipo=" + codEquipo + ";");

        while (res.next()) {
            int c = res.getInt( columnIndex: 1);
            sum = sum + c;
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }

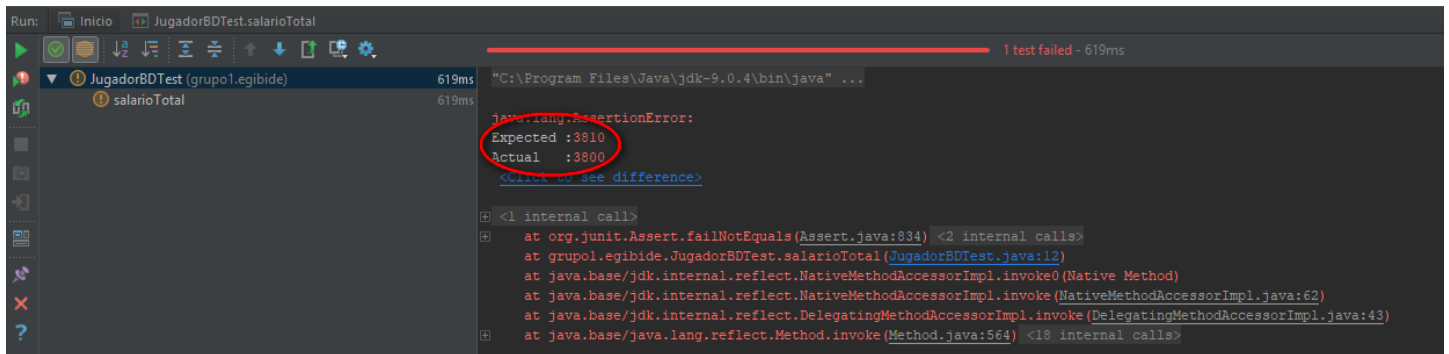
    GestorBD.desconectar();
    return sum;
}
```

2. En la clase de pruebas **JugadorBDTest** implementamos la comprobación:



3. En una consulta previa a la BBDD hemos comprobado que en este caso, la suma de los salarios del equipo 2 es de 3800€.

Si indicamos que esperamos que la suma sea 3810, la comprobación nos indica que el parámetro **no se corresponde** con el que devuelve el método.



```
Run: Inicio JugadorBDTest.salarioTotal
JugadorBDTest (grupo1.egibide) 619ms
salarioTotal 619ms
1 test failed - 619ms

java.lang.AssertionError:
Expected :3810
Actual   :3800
<Click to see difference>

<1 internal call>
at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>
at grupo1.egibide.JugadorBDTest.salarioTotal(JugadorBDTest.java:12)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:564) <18 internal calls>
```

ii. contarJugador() en EquipoBD:

1. Este método cuenta los jugadores que hay en un equipo determinado que el jugador proporciona como parámetro (codEquipo).

```
public static List<Jugador> buscarEquipos(int codigo) {
    Jugador JUGADOR = null;
    List<Jugador> lista = new ArrayList<>();
    Connection conexion = GestorBD.conectar();

    try {
        Statement st = conexion.createStatement();
        String sql = "SELECT * FROM Jugador where Equipo_codEquipo = " + codigo;
        ResultSet rs = st.executeQuery(sql);

        while (rs.next()) {
            /* JUGADOR = new Jugador(
                rs.getInt("codJugador"),
                rs.getString("Nombre"),
                rs.getString("Nick"),
                rs.getInt("Salario"),
                rs.getString("FechaAlta"),
                rs.getString("Posicion"),
                rs.getInt("Equipo_codEquipo")
            ); */
            lista.add(
                new Jugador(
                    rs.getInt( columnLabel: "codJugador"),
                    rs.getString( columnLabel: "Nombre"),
                    rs.getString( columnLabel: "Nick"),
                    rs.getInt( columnLabel: "Salario"),
                    rs.getString( columnLabel: "FechaAlta"),
                    rs.getString( columnLabel: "Posicion"),
                    rs.getInt( columnLabel: "Equipo_codEquipo")
                ));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    GestorBD.desconectar();
    return lista;
}
```

2. Si hacemos una consulta a la BBDD, vemos que en el equipo 1 hay 4 Jugadores.

1 • `SELECT * FROM Jugador WHERE Equipo_codEquipo=1;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

	codJugador	nombre	nick	salario	fechaAlta	Posicion	Equipo_codEquipo
7	NULL	SARIT96	800	15-02-97		1	1
8	NULL	dierouu	20000	125420	1	1	1
12	NULL	pepii	1500	15-05-18	ala	1	1
13	NULL	pepto5	2000	15-05-17	ala	1	1
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. En la comprobación de la clase EquipoBD, llamada EquipoBDTest indicamos que esperamos que el resultado del método **contarJugador()** sea 4, por lo que la comprobación se realiza correctamente.

```

@Test
public void contarJugador() { //Cuenta los jugadores que hay en un equipo
    assertEquals("expected: 4, EquipoBD.contarJugador( codEquipo: 1))",
        4, EquipoBD.contarJugador(1));
}

@Test
public void buscarEquipos() {
}
    
```

JUnit Test Results:

Test Name	Duration	Status
EquipoBDTest (grupo1.egibide)	702ms	Passed
contarJugador	702ms	Passed

Process finished with exit code 0

4. Si por el contrario indicamos que esperamos que haya 3 Jugadores, JUnit nos va a indicar que el test ha fallado porque se han encontrado 4 Jugadores.

```

package grupo1.egibide;

import ...

public class EquipoBDTest {

    @Test
    public void contarJugador() { //Cuenta los jugadores que hay en un equipo
        assertEquals("expected: 3, EquipoBD.contarJugador( codEquipo: 1))",
            3, EquipoBD.contarJugador(1));
    }
}
    
```

JUnit Test Results:

Test Name	Duration	Status
EquipoBDTest (grupo1.egibide)	692ms	Failed
contarJugador	692ms	Failed

Test Failure Details:

```

java.lang.AssertionError:
Expected :3
Actual   :4
    
```

Click to see difference