Argentina programa

Desarrollo de aplicaciones JAVA

Guía V

"Proyecto Transversal: Gestión Universidad"

En la guía anterior, aprendimos cómo establecer una conexión desde una aplicación Java hacia una base de datos relacional utilizando el framework JDBC, y realizar operaciones sobre ella. Ahora, en esta nueva guía, daremos un paso más allá al explorar las mejores prácticas de la programación orientada a objetos para estructurar nuestro proyecto. Construiremos una aplicación completa en la cual el usuario interactuará a través de una interfaz gráfica, brindándole una experiencia más intuitiva y amigable.

Es importante que repartan las tareas entre los miembros del equipo, de tal forma que: uno crea repositorio en GitHub, agrega a sus compañeros como colaboradores y luego los demás clonan, y cada uno hace una de las clases que forman parte del proyecto y suben al repositorio los cambios.

Sistema de gestión para la Universidad de La Punta:

La Universidad de La Punta cree necesario utilizar un sistema para poder llevar el registro de los alumnos de la institución y las materias que se dictan en la misma. Adicionalmente se necesita poder registrar las materias que cursa cada alumno. El sistema debe permitir cargar la calificación obtenida (nota) cuando un alumno rinde un examen final. Para cada materia que cursa un alumno solo se registrará la última calificación obtenida, o sea no se mantiene registro de las notas obtenidas anteriormente, por lo que, si un alumno rinde el examen final de una materia y obtiene una calificación de "2", y luego rinde nuevamente el examen para la materia y obtiene una calificación de "9" solo quedará registro de esta última.

Funcionalidad: el sistema deberá

1. Permitir al personal administrativo listar las materias que cursa un alumno.
2. Permitir al personal administrativo listar los alumnos inscriptos en una determinada materia.
3. Permitir que un alumno se pueda inscribir o des-inscribir en las materias que desee.
4. Permitir registrar la calificación final de una materia que está cursando un alumno.
5. Permitir el alta, baja y modificación de los alumnos y las materias.

Modelo de BD sugerido.

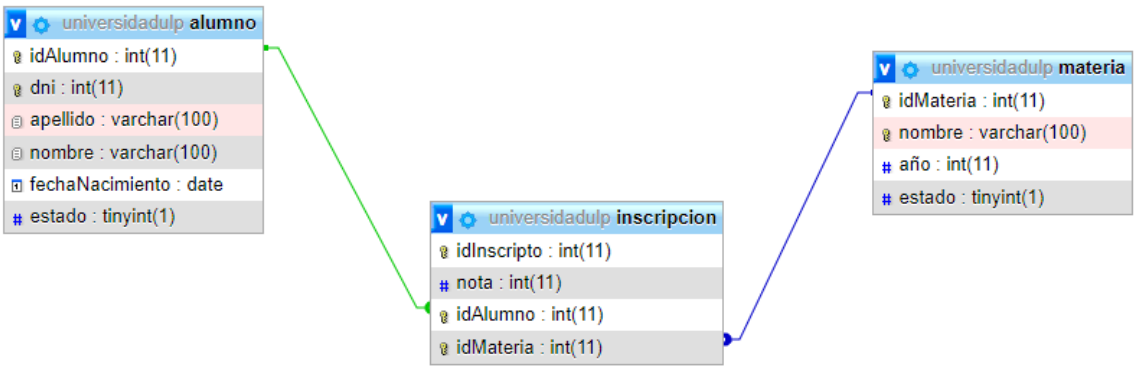


Ilustración 1 Modelo Relacional Universidad

DESARROLLO

Para implementar la solución a esta problemática, en primer lugar, utilizaremos el mismo proyecto creado en la sección anterior "universidadEjemplo" y vamos a aplicar el patrón MVC (Modelo-Vista- Controlador) en dónde agruparemos las clases que vamos a utilizar en paquetes de acuerdo a su responsabilidad (tarea), es decir, por un lado las clases que van a representar a nuestro modelo de datos, por otro, las clases que van a actuar como intermediarias entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno y por último, las clases que representarán las interfaz gráfica a través de las cuales los usuarios interactuaran con nuestra aplicación como: menús, formularios, entre otros.

Estructura del Proyecto:

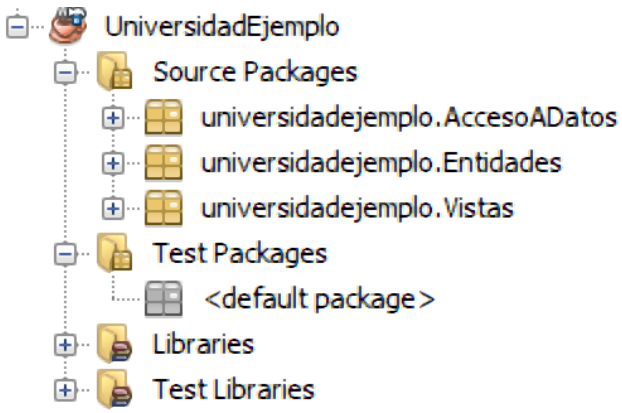


Ilustración 2 "Estructura de proyecto java"

Dentro del paquete principal de nuestro proyecto, crearemos 3(tres) paquetes de nombre:

-entidades

-accesoADatos

-vistas

Como se muestra en la *"ilustración 2"*.

CLASES ENTIDAD:

Para que desde el lenguaje de programación Java se pueda tratar **cada registro o fila de las tablas como un objeto Java**, se debe crear una clase asociada a cada tabla de la base de datos y las relaciones entre ellas. A ese tipo de clases se las conoce como **Clases Entidad (***Entity Classes***)**.

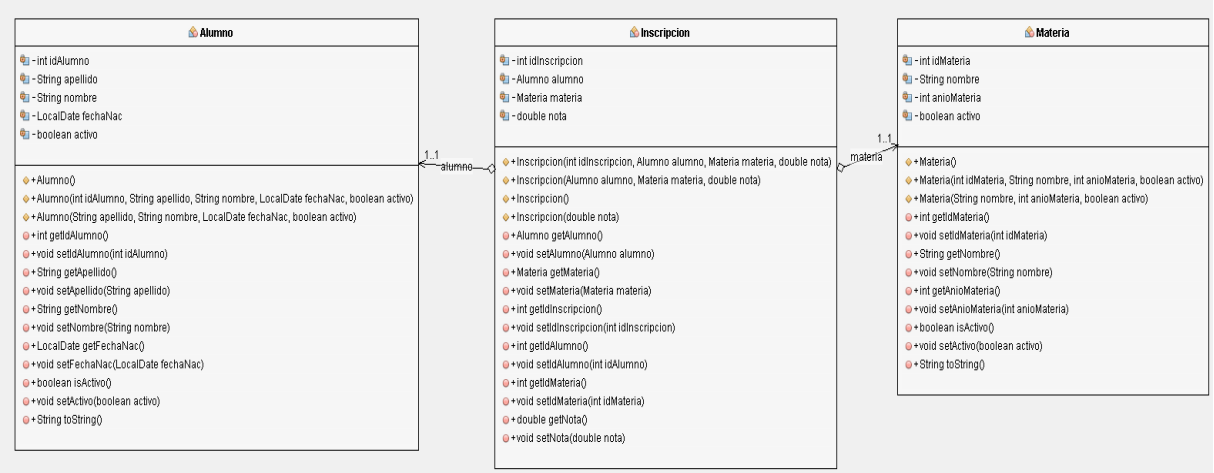


Ilustración 3 Clases entidades

Como se puede observar en el diagrama UML cada clase entidad tiene los mismos atributos que las tablas de la base de datos, y 3 constructores: uno vacío, uno con todos los atributos incluido el id y otro con todos los atributos sin incluir el id; otros miembros adicionales de las clases serán los métodos getter y setter; además de las relaciones entre ellas plasmadas como asociaciones, es decir, la tabla Inscripción tendrá un atributo de tipo Materia y otro de tipo Alumno; lo que en la base de datos eran claves foráneas, aquí son asociaciones.

ESTABLECIENDO LA CONEXIÓN A LA BASE DE DATOS:

Dentro de nuestro proyecto crearemos una clase especial que será no sólo la responsable de cargar los drivers de conexión al gestor de base de datos MySQL, sino también la de establecer la conexión a la base de datos que vamos a utilizar; empleando las clases vistas en la guía anterior.

Se muestra a continuación el código de la clase "Conexión.java" y la explicación.

C:/Users/Usuario/Documents/GitHub/EjemploUniversidad/src/universidadejemplo/AccesoADatos/Conexion.java

```
package universidadejemplo.AccesoADatos;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;



public class Conexion {

    private static final String URL="jdbc:mysql://localhost/";
    private static final String DB="universidadulp";
    private static final String USUARIO="root";
    private static String PASSWORD="";

    private static Connection connection;


    //Metodo constructor

    private Conexion(){}


public static Connection getConexion(){


    if (connection == null) {

        try {
            Class.forName("org.mariadb.jdbc.Driver");
            // Setup the connection with the DB
            connection = DriverManager
                    .getConnection(URL+DB + "?useLegacyDatetimeCode=false&serverTimezone=UTC"
                            + "&user=" + USUARIO + "&password=" + PASSWORD);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Error al conectarse a la BD "+ex.getMessage());
        }catch(ClassNotFoundException ex){

            JOptionPane.showMessageDialog(null, "Error al cargar los Drivers "+ex.getMessage());
        }
    }
    return connection;
    }


}
```

1.1 of 1 2023.05.23 08:43:57

```
13  public class Conexion {
14
15      private static final String URL="jdbc:mysql://localhost/";
16      private static final String DB="universidadulp";
17      private static final String USUARIO="root";
18      private static final String PASSWORD="";
19
20      private static Connection connection;
```

Como se en la imagen, desde la línea 15 hasta las 18, están declaradas una serie de constantes de tipo String con información que utilizaremos para la conexión.

En la constante URL la cadena establece que utilizaremos un conexión utilizando jdbc a una base de datos MySql y que se encuentra disponible en la misma PC (localhost); si la base de datos se encontrara en otro host, reemplazaríamos localhost por el ip o nombre del equipo en dónde se encuentra.

La constante DB contiene el nombre de la base de datos a la que vamos a querer acceder.

Cuando instalamos XAMPP, al agregarnos el gestor MYSQL, establece una configuración por defecto, en donde el usuario de acceso a la base de datos es “root” sin contraseña. Eso es lo que está definido en las constantes USUARIO y PASSWORD.

En la línea 20, la constante “connection” representará a un objeto de tipo Connection a través del cual podremos enviar nuestras peticiones a la Base de Datos.

```
22
23      //Metodo constructor
24
25      private Conexion(){}
26
27
```

La Clase Conexion, tiene un único constructor, pero es privado, como se observa en la línea 25. La consecuencia de tener un único constructor y encima privado, es que no nos permitirá crear instancias de la clase Conexion. Esto nos posibilitará como lo van a ver más adelante en el código, tener un único objeto Connection activo durante la ejecución de nuestra aplicación.

```
28  public static Connection getConexion(){
29
30      if (connection == null) {
31          try {
32              Class.forName("org.mariadb.jdbc.Driver");
33              // Setup the connection with the DB
34              connection = DriverManager
35                      .getConnection(URL+DB + "?useLegacyDatetimeCode=false&serverTimezone=UTC"
36                              + "&user=" + USUARIO + "&password=" + PASSWORD);
37
38          } catch (SQLException ex) {
39              JOptionPane.showMessageDialog(null, "Error al conectarse a la BD "+ex.getMessage());
40          }catch(ClassNotFoundException ex){
41
42              JOptionPane.showMessageDialog(null, "Error al cargar los Drivers "+ex.getMessage());
43          }
44      }
45      return connection;
46      }
47  }
```

La Clase Conexión tiene un único método público encargado de retornar un objeto de tipo Connection, este método tiene que ser static para que pueda ser invocado desde afuera de la clase sin necesitad crear una instancia de la misma, ya que no podríamos por el hecho de que tiene un único constructor privado.

En la línea 30 verifica si el atributo static “connection” es nulo, si es así significa que nadie ha llamado este método desde la aplicación para establecer una conexión a la base de datos (recordemos que el objetivo es mantener un único objeto Connection activo durante el ciclo de vida de nuestra aplicación)

En la línea 32 se cargan los driver de conexión al gestor de base de datos; previamente tuvimos que incorporar a nuestro proyecto las librerías de MariaDB o MYSQL como se explicó en la guía anterior. La invocación Class.forName() podría lanzar la ClassNotFoundException si no cargamos previamente dicha librería.

En la línea 34 se establece la conexión a la base de datos invocando al método getConnection() de la clase DriverManager pasando por parámetro un String que contiene la URL de la base de datos, PASSWORD y CONTRASEÑA como se indica a continuación.

DriverManager.getConnection(URL+DB + "?useLegacyDatetimeCode=false&serverTimezone=UTC" + "&user=" + USUARIO + "&password=" + PASSWORD);

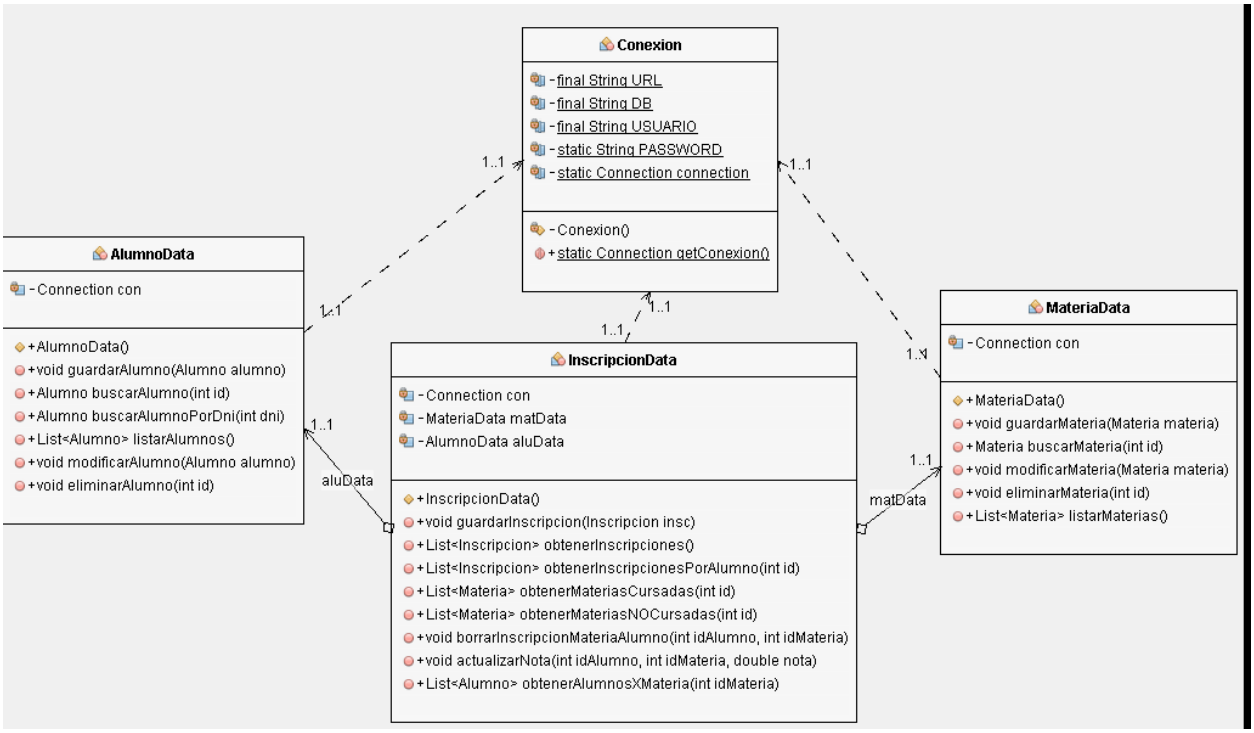
El método getConnection() puede lanzar la SQLException si la cadena que le pasamos como parámetro es incorrecta, por ejemplo el nombre de la base de datos no existe; excepción que capturamos en la línea 38, que de producirse se informará a través de un diálogo al usuario.

Al finalizar, en la línea 45, retornamos el objeto Connection.

Quiero aclarar, que la forma en como está implementada la clase Conexión es una sugerencia, cada uno de ustedes puede optar por su propia implementación; sólo deberán tener en cuenta que para poder establecer una conexión a la base de datos, primero se deben cargar los driver y luego establecer la conexión a través de la invocación del método getConnection().

CLASES DE ACCESO A DATOS:

Estas clases se encargan de la persistencia de los datos y su acceso. En este proyecto utilizando tecnología JDBC (Java Database Connectivity) para interactuar con la base de datos. Proporcionan métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos. Estas clases son: 'AlumnoData', 'MateriaData' e 'InscripcionData' y podríamos incluir aquí también una clase adicional que se encargue de cargar los driver y conexión a la base de datos.



A continuación detallaremos la implementación de AlumnoData para que en base a este ejemplo usted pueda implementar MateriaData y CursadaData; obviamente con ayuda de sus compañeros de equipo, los videos de soporte y aporte de su docente tutor.

Como mencionamos anteriormente, estas clases se van encargar tanto de la persistencia de datos como del acceso a los mismos; por lo tanto, en sus métodos valiéndonos de una **Connection** vamos a generar los **PreparedStatement** con las sentencias SQL que vamos a necesitar enviar al gestor de Bases de Datos.

Implementación sugerida para AlumnoData

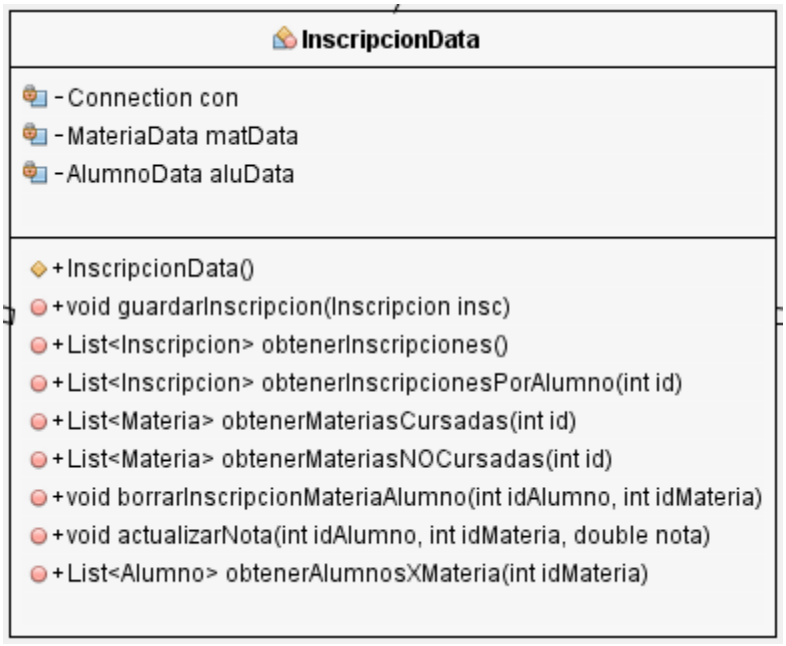
```java
package universidadejemplo.AccesoADatos;

import universidadejemplo.Entidades.Alumno;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;

public class AlumnoData {

    private Connection con = null;

  public AlumnoData() {

            con = Conexion.getConexion();
  }

    public void guardarAlumno(Alumno alumno) {

        String sql = "INSERT INTO alumno (dni, apellido, nombre, fechaNacimiento, estado) VALUES (?, ?, ?, ?, ?)";
        try {
            PreparedStatement ps = con.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            ps.setInt(1, alumno.getDni());
            ps.setString(2, alumno.getApellido());
            ps.setString(3, alumno.getNombre());
            ps.setDate(4, Date.valueOf(alumno.getFechaNacimiento()));//localDate a Date
            ps.setBoolean(5, alumno.isEstado()); // if reducido
            ps.executeUpdate();
            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                alumno.setIdAlumno(rs.getInt("idAlumno"));
            JOptionPane.showMessageDialog(null, "Alumno añadido con exito."); 38
            }
            ps.close();

        } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, "Error al acceder a la tabla Alumno"+ex.getMessage()); 43
        }

    }
    public Alumno buscarAlumno(int id) {
        Alumno alumno = null;
        String sql = "SELECT  dni, apellido, nombre, fechaNacimiento FROM alumno WHERE idAlumno = ? AND estado = 1";
        PreparedStatement ps = null;
        try {
            ps = con.prepareStatement(sql);
            ps.setInt(1,id );

            ResultSet rs = ps.executeQuery();

            if (rs.next()) {
                alumno=new Alumno();
                alumno.setIdAlumno(id);
                alumno.setDni(rs.getInt("dni"));
                alumno.setApellido(rs.getString("apellido"));
                alumno.setNombre(rs.getString("nombre"));
                alumno.setFechaNacimiento(rs.getDate("fechaNacimiento").toLocalDate());
                alumno.setEstado(true);


            } else {
                JOptionPane.showMessageDialog(null, "No existe el alumno");

```

```
68              ps.close();
69          } catch (SQLException ex) {
70          JOptionPane.showMessageDialog(null, "Error al acceder a la tabla Alumno "+ex.getMessage()); 71
            }
72
73          return alumno;
74      }
75
76      public Alumno buscarAlumnoPorDni(int dni) {
77          Alumno alumno = null;
78          String sql = "SELECT idAlumno, dni, apellido, nombre, fechaNacimiento FROM alumno WHERE dni=? AND estado = 1";
79          PreparedStatement ps = null;
80          try {
81              ps = con.prepareStatement(sql);
82              ps.setInt(1,dni );
83              ResultSet rs = ps.executeQuery();
84
85              if (rs.next()) {
86                  alumno=new Alumno();
87                  alumno.setIdAlumno(rs.getInt("idAlumno"));
88                  alumno.setDni(rs.getInt("dni"));
89                  alumno.setApellido(rs.getString("apellido"));
90                  alumno.setNombre(rs.getString("nombre"));
91                  alumno.setFechaNacimiento(rs.getDate("fechaNacimiento").toLocalDate());
92                  alumno.setEstado(true);
93
94
95              } else {
96              JOptionPane.showMessageDialog(null, "No existe el alumno"); 97
                }
98              ps.close();
99          } catch (SQLException ex) {
100             JOptionPane.showMessageDialog(null, "Error al acceder a la tabla Alumno "+ex.getMessage());
101         }
102
103
104         return alumno;

105     }
106
107     public List<Alumno> listarAlumnos() {
108
109         List<Alumno> alumnos = new ArrayList<>();
110         try {
111             String sql = "SELECT * FROM alumno WHERE estado = 1 ";
112             PreparedStatement ps = con.prepareStatement(sql);
113             ResultSet rs = ps.executeQuery();
114             while (rs.next()) {
115                 Alumno alumno = new Alumno();
116
117                 alumno.setIdAlumno(rs.getInt("idAlumno"));
118                 alumno.setDni(rs.getInt("dni"));
119                 alumno.setApellido(rs.getString("apellido"));
120                 alumno.setNombre(rs.getString("nombre"));
121                 alumno.setFechaNacimiento(rs.getDate("fechaNacimiento").toLocalDate());
122                 alumno.setEstado(rs.getBoolean("estado"));
123                 alumnos.add(alumno);
124             }
125             ps.close();
126
127
128         } catch (SQLException ex) {
129             JOptionPane.showMessageDialog(null, " Error al acceder a la tabla Alumno "+ex.getMessage());
130         }
131         return alumnos;
132     }
133
```

```
134     public void modificarAlumno(Alumno alumno)
135
136         String sql = "UPDATE alumno SET dni = ? , apellido = ?, nombre = ?, fechaNacimiento = ? WHERE  idAlumno = ?";
137         PreparedStatement ps = null;
138
139         try {
140             ps = con.prepareStatement(sql);
141             ps.setInt(1, alumno.getDni());
142             ps.setString(2, alumno.getApellido());
143             ps.setString(3, alumno.getNombre());
144             ps.setDate(4, Date.valueOf(alumno.getFechaNacimiento()));
145             ps.setInt(5, alumno.getIdAlumno());
146             int exito = ps.executeUpdate();
147
148             if (exito == 1) {
149                 JOptionPane.showMessageDialog(null, "Modificado Exitosamente.");
150             } else {
151                 JOptionPane.showMessageDialog(null, "El alumno no existe");
152             }
153
154         } catch (SQLException ex) {
155             JOptionPane.showMessageDialog(null, "Error al acceder a la tabla Alumno "+ex.getMessage());
156         }

157
158     }
159
160
161
162     public void eliminarAlumno(int id) {
163
164         try {
165             String sql = "UPDATE alumno SET estado = 0 WHERE idAlumno = ? ";
166             PreparedStatement ps = con.prepareStatement(sql);
167             ps.setInt(1, id);
168             int fila=ps.executeUpdate();
169
170             if(fila==1){
171                 JOptionPane.showMessageDialog(null, " Se eliminó el alumno.");
172             }
173               ps.close();
174         } catch (SQLException e) {
175             JOptionPane.showMessageDialog(null, " Error al acceder a la tabla Alumno");
176         }
177     }
178
179
180 }
```

CLASE DE ACCESO A DATOS DE INSCRIPCIONES

(InscripcionData)



Esta clase además de poseer los métodos que permitirán hacer un CRUD sobre la tabla Inscripcion, tiene métodos que permiten hacer operaciones un poco más complejas como el método "obtenerMateriasCursadas", que recibiendo el id del alumno, nos retornará las materias en las que este está inscripto. La elección de mostrar en detalle la implementación de este método en particular es la suposición de que explicando cómo está construido, usted será capaz de implementar el resto de los métodos de la clase InscripcionData, con ayuda de este material escrito, los videos, debate entre los compañeros de equipo y por supuesto la colaboración de su tutor de clase.

```
114    public List<Materia> obtenerMateriasCursadas(int id){
           List<Materia> materias = new ArrayList<Materia>();
116
117        try {
118            String sql = "SELECT inscripcion.idMateria, nombre, año FROM inscripcion,"
119                    + " materia WHERE inscripcion.idMateria = materia.idMateria\n" +
120                "AND inscripcion.idAlumno = ?;";
121            PreparedStatement ps = con.prepareStatement(sql);
122            ps.setInt(1, id);
123            ResultSet rs = ps.executeQuery();
124            Materia materia;
125            while(rs.next()){
126                materia = new Materia();
127                materia.setIdMateria(rs.getInt("idMateria"));
128                materia.setNombre(rs.getString("nombre"));
129                materia.setAnio(rs.getInt("año"));
130                materias.add(materia);
131            }
132            ps.close();
133        } catch (SQLException ex) {
134            JOptionPane.showMessageDialog(null, "Error al obtener Inscripciones."+ex.getMessage());
135        }
136        return materias;
137    }
```

Como podemos observar en la declaración del método en la línea 114, recibe como parámetro el id de un alumno y retornará una lista de las materias en las que está inscripto (List<Materia>). Por lo tanto, en la línea 115 creamos una lista vacía de materias que luego iremos llenando.

La sentencia que enviaremos a la base de datos será un select en el que solicitaremos los datos de las materias en las que está inscripto el alumno cuyo id recibimos por parámetro; como puede usted observar en la línea 118.

String **sql** = "SELECT inscripcion.idMateria, nombre, año FROM inscripción JOIN materia ON(inscripción.idMateria=materia.idMateria) WHERE inscripcion.idAlumno = ?;"

SELECT inscripción.idMateria, nombre, año

En la cláusula SELECT indicamos que vamos a proyectar los campos de la tabla Materia: idMateria, nombre y año; como en la cláusula FROM vamos a reunir dos tablas: INSCRIPCION y MATERIA tendríamos un inconveniente con aquellos campos que tenga el mismo nombre, como ocurre con idMateria, que figura tanto en la tabla INSCIPCION como MATERIA; por lo tanto en el SELECT nos vemos obligados para evitar un error de ambigüedad de nombre cualificar el campo idMateria con alguna de las tablas, en este caso elegía INSCRIPCION, es por ello que figura en el SELECT como **inscripción.idMateria**, aunque también podría haberlo escrito **materia.idMateria.**

FROM inscripción JOIN materia ON (inscripción.idMateria=materia.idMateria)

Para poder proyectar en la cláusula SELECT los datos de las materias, pero sólo de aquellas en las que está inscripto un determinado alumno, y sabemos que en la tabla inscripción solo tenemos el id del alumno y el id de la materia en el que éste está inscripto, por lo tanto si necesito completar la información con el resto de los datos de la materia que se corresponden con ese id, debo reunir (JOIN) las tablas INSCRIPCION y MATERIA, a través de clave primaria de Materia con la clave foránea idMateria en Inscripcion; es lo que indicamos en la cláusula ON(inscripción.idMateria=materia.idMateri).

WHERE inscripción.idAlumno = ?

En la cláusula Where colocaremos como condición de filtrado que solo queremos las filas en las que el idAlumno de la tabla Inscripcion sea un dato que reemplazaremos dinámicamente (?).

Ahora, para poder enviar esta sentencia a la base de datos, creamos un objeto del tipo PreparedStatement utilizando el objeto Connection (referenciado con la variable atributo con).

PreparedStatement ps = con.prepareStatement(sql);

Como la sentencia que pretendemos enviar, tiene un parámetro dinámico (?) después de la cláusula Where, que representa el valor entero del id de un alumno, lo setearemos de la siguiente forma.

ps.setInt(1, id);

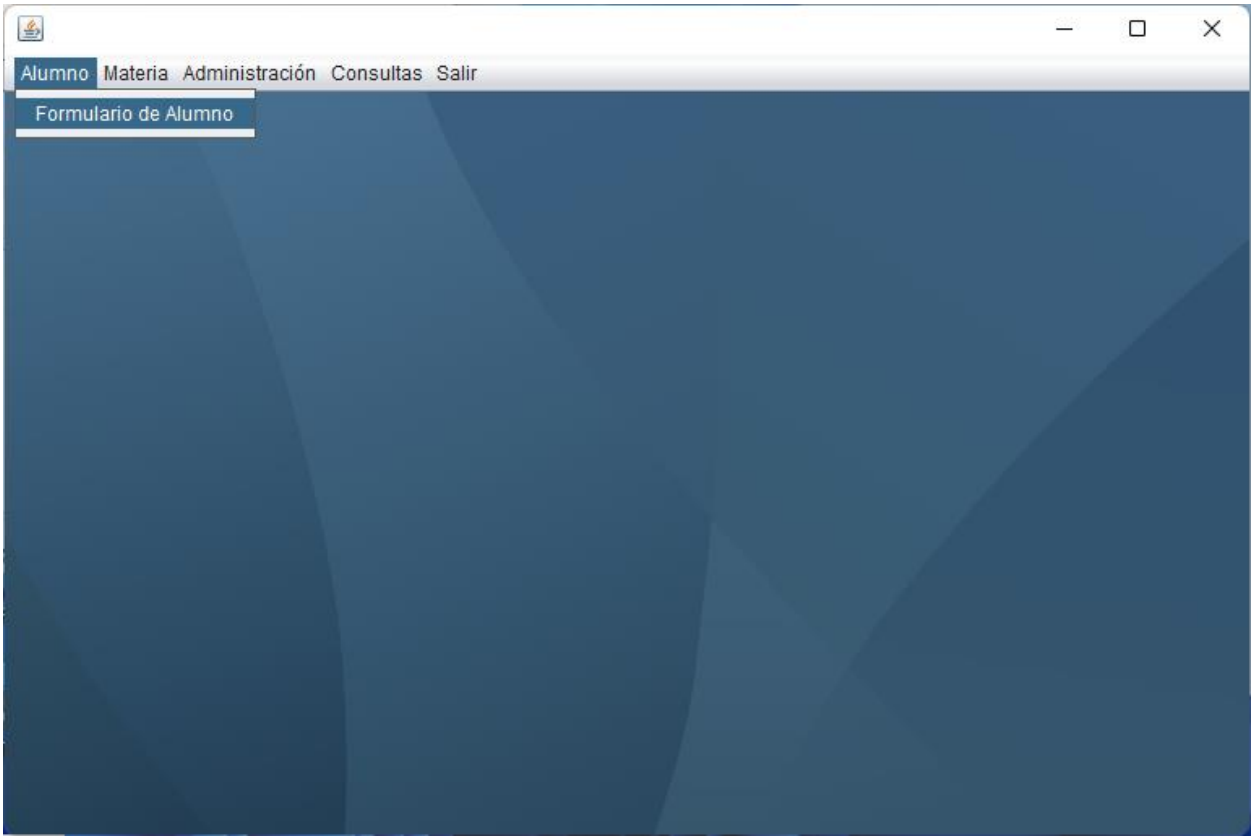
Indicando al objeto PreparedStatement referenciado por la variable **ps,** a través del método setInt que el primer y único parámetro dinámico que tenemos en la sentencia sql, es un entero y que vamos a reemplazar ese signo ? por el id del alumno guardado en la variable entera id.

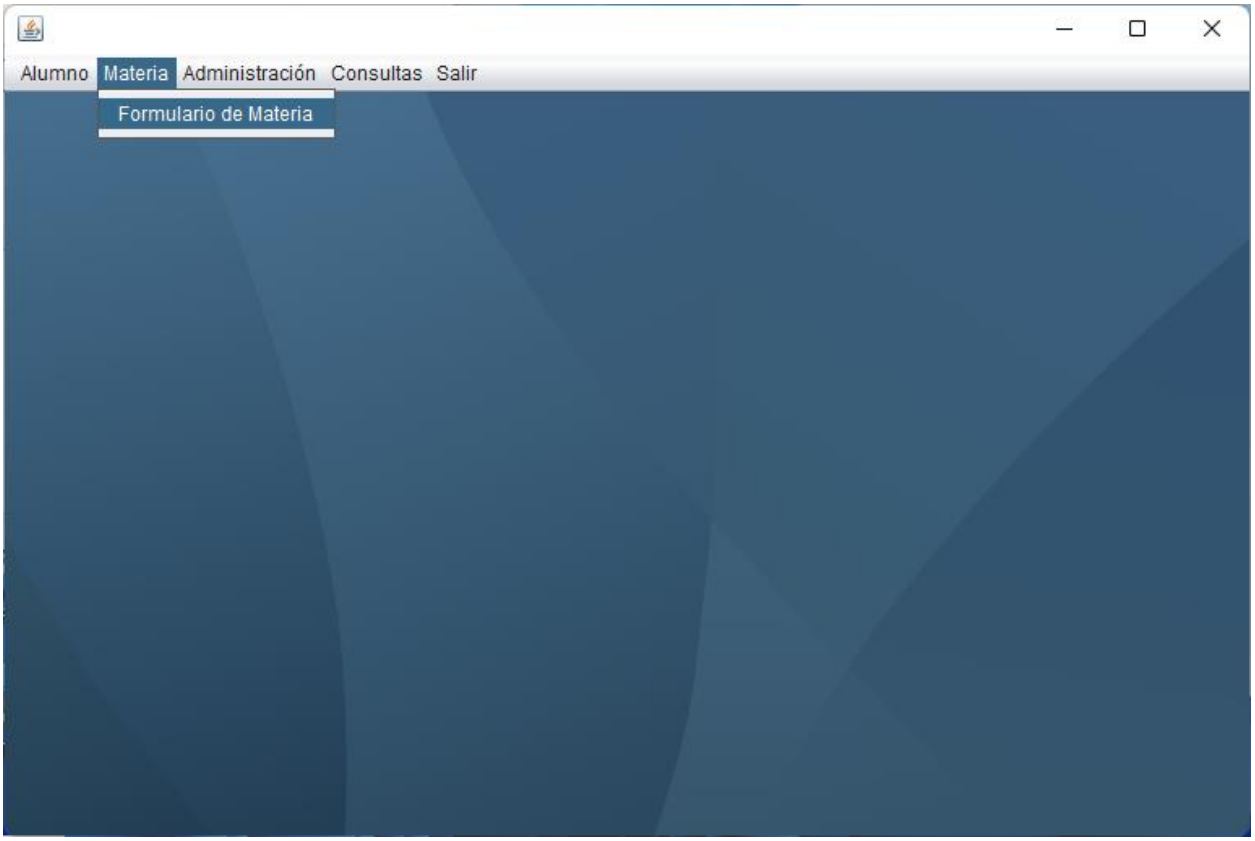
Una vez seteado el parámetro dinámico, enviaremos la sentencia a la base de datos utilizando el método executeQuery que nos devolverá un ResultSet que contendrá las Materias en las que el alumno está inscripto. En la guía anterior se explicó que representa un ResultSet, si no lo recuerda puede volver a leer dicha explicación.

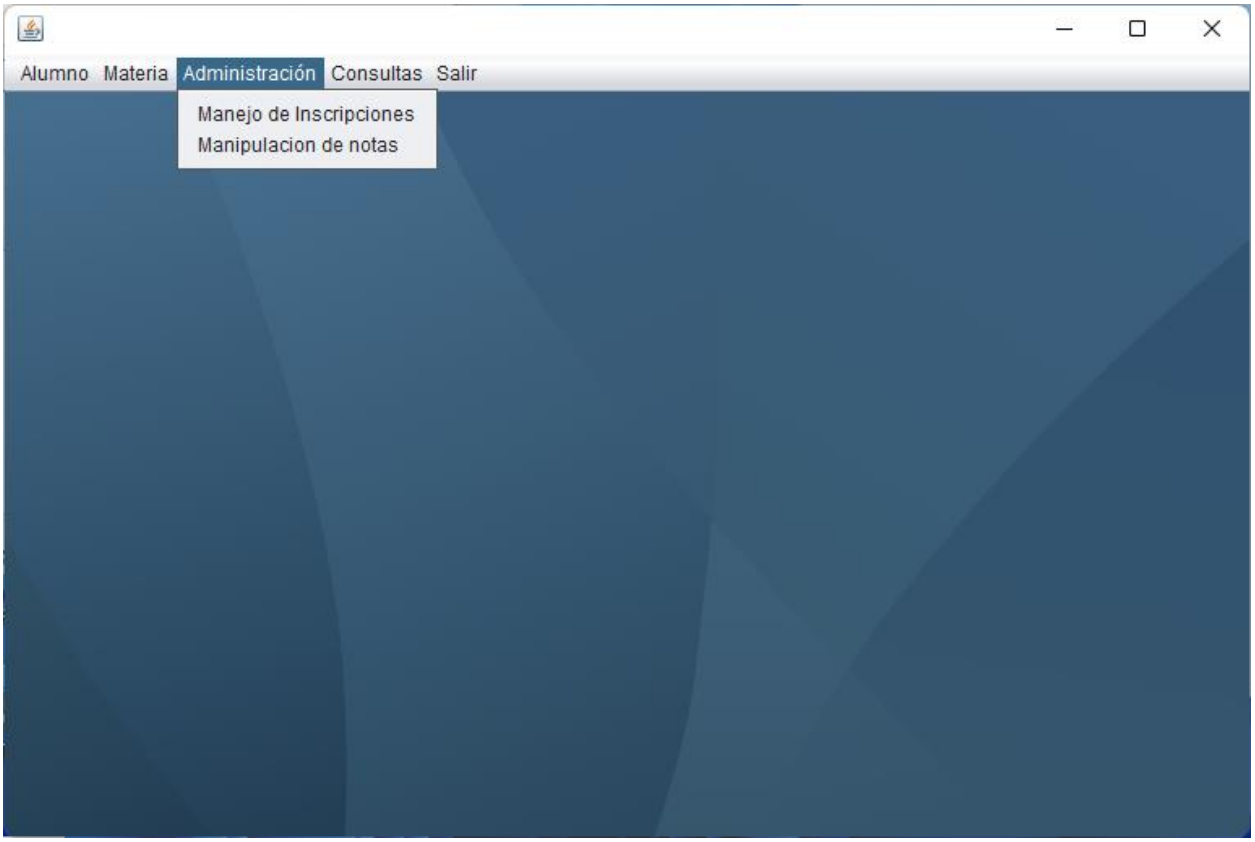
ResultSet rs = ps.executeQuery();

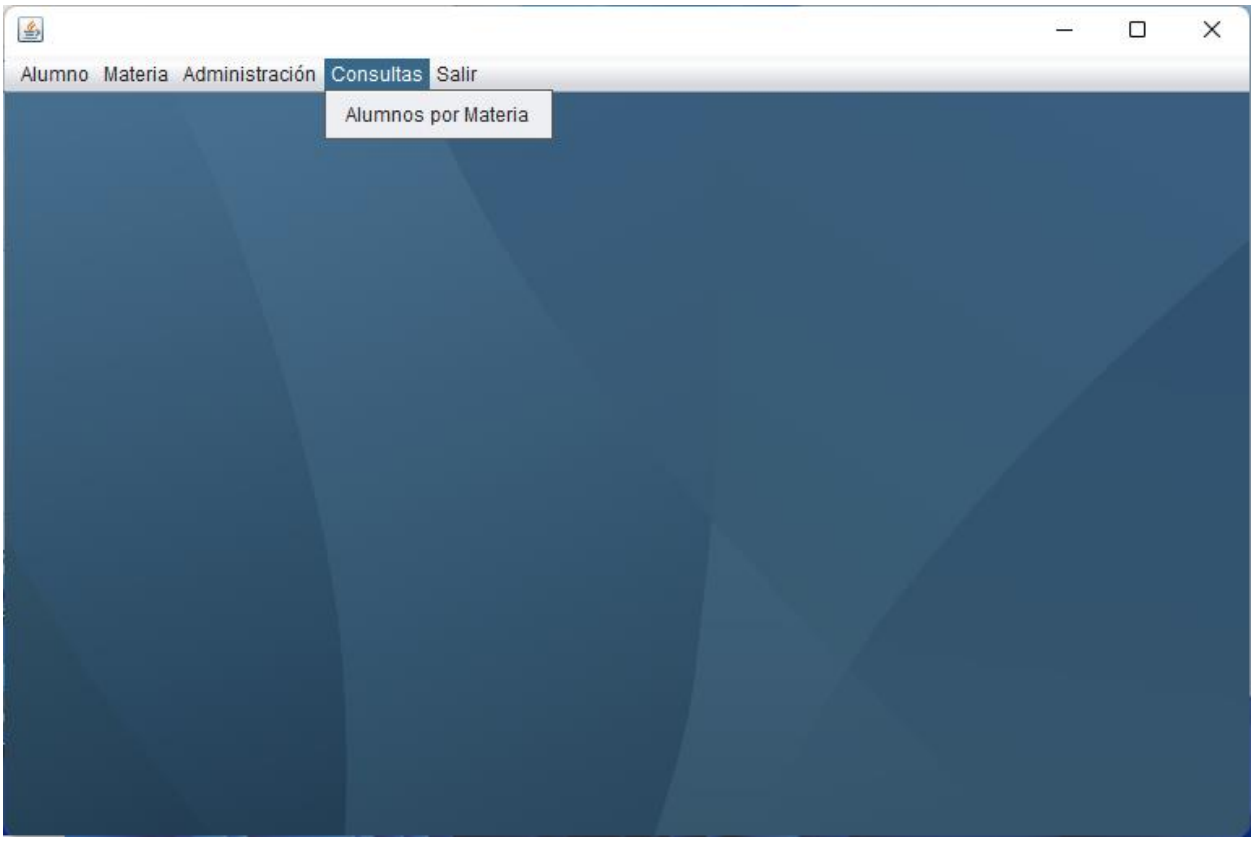
VISTAS SUGERIDAS:

Menú Principal: Construido con un JFrame, JMenuBar, JMenu y JDesktopPane dentro del cual se visualizaran las JInternalFrame que corresponda a cada vista elegida por el usuario.



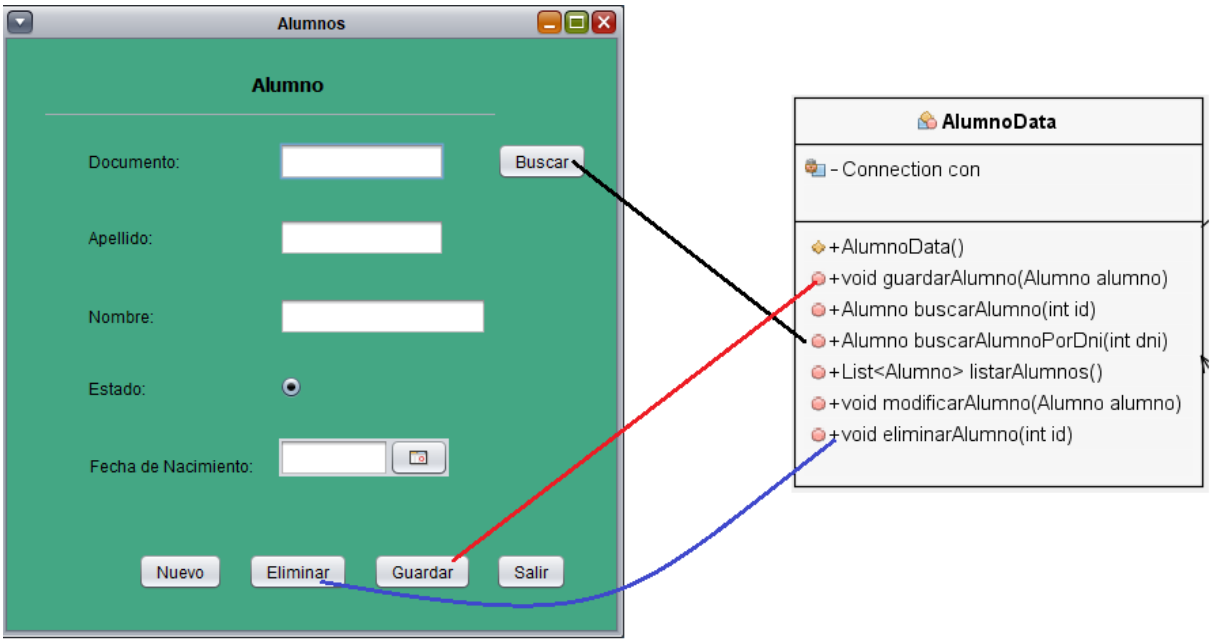



Alumno
Materia
Administración
Consultas
Salir
Manejo de Inscripciones
Manipulacion de notas

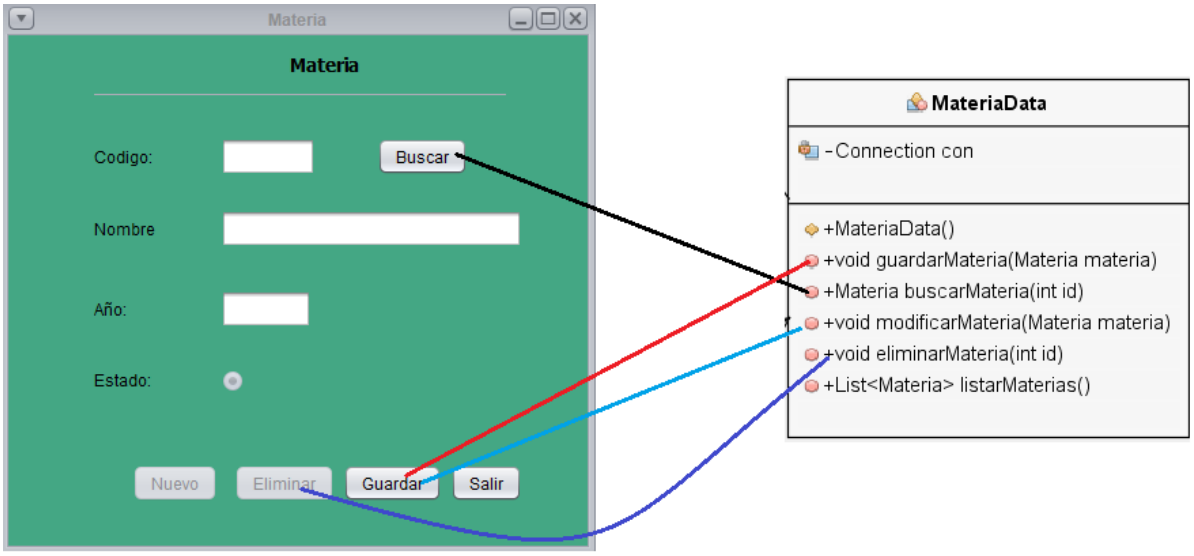

Alumno
Materia
Administración
Consultas
Salir
Alumnos por Materia

A continuación se detallan cada una de las vistas y la clase de acceso a datos vinculada a dicha vista.

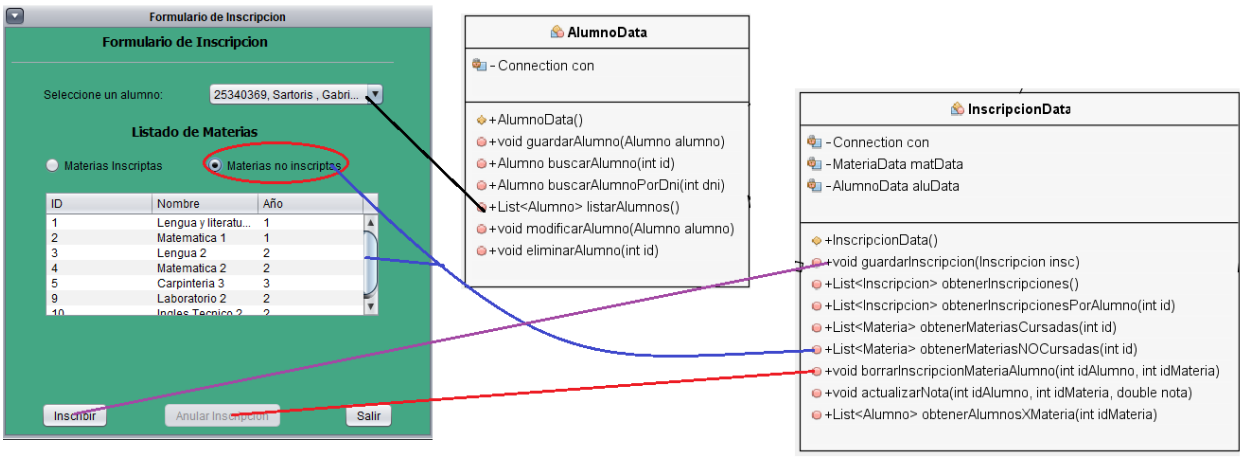
Gestión de Alumnos

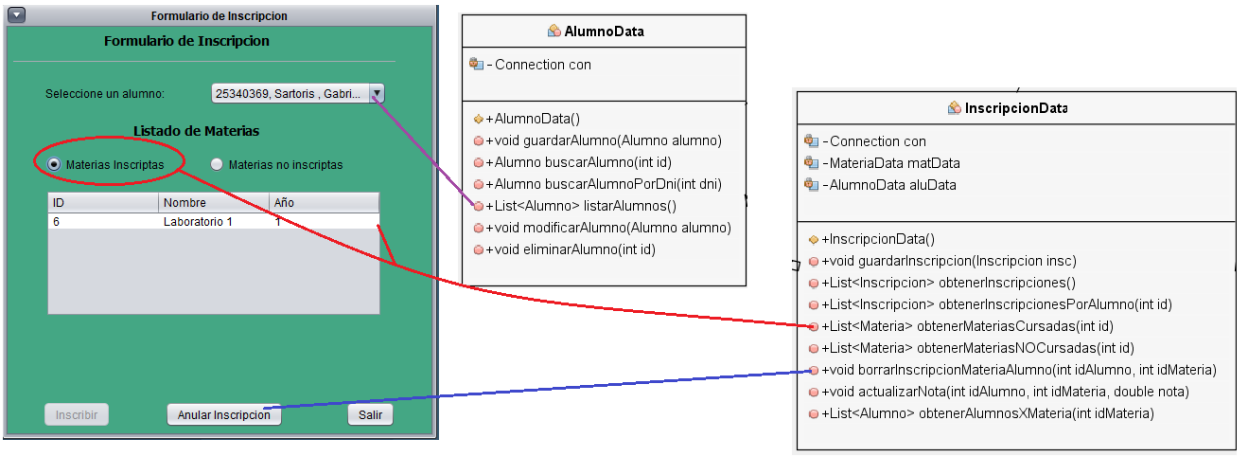


Gestión de Materias:

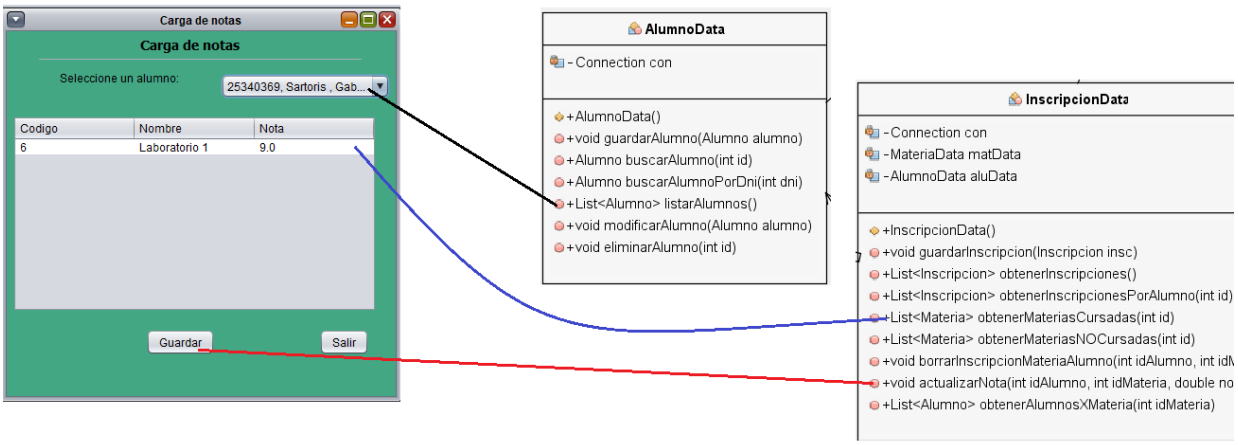


Inscripciones:





Actualización de notas:



Consulta de Alumnos por materia.