UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	Abril 2023 – Octubre 2023
ASIGNATURA	:	POO
TEMA	:	Laboratorio 1
NOMBRES	:	Kelvin Quezada
NIVEL-PARALELO	:	Segundo “A”
DOCENTE	:	Ing. Cevallos Farias Javier Moyota
FECHA DE ENTREGA	:	04/08/2023

SANTO DOMINGO - ECUADOR

2023

Contenido

1. Introducción

En la programación orientada a objetos es el proceso de la programación que implica la creación de clases que describen las características y el comportamiento de los objetos y la creación de instancias de esas clases para trabajar con objetos concretos. Los objetos interactúan entre sí mediante el uso de sus métodos y el acceso a sus atributos.

Mediante este tema pudimos desarrollar proyectos por medio de interfaces que nos ayudan al registro de usuario y tener un formulario con la base de datos que se guardara en la nube dicha información ingresada.

2. Objetivos

2.1.1. Objetivos Generales

- Realizar un programa que ayude al registro de datos y que se aguarde la información en una base de datos.

2.1.2. Objetivos Específicos

- Realizar este ejercicio con interfaces con caja de textos, combo box y una caja de Tabla para almacenar la información.

3. Marco Teórico

3.1. Programación Orientado a Objeto

Según (Canelo, 2020) la programación orientada a objetos (POO) es un paradigma de programación, es decir, un patrón o estilo de programación que nos dice cómo usarlo. Se basa en los conceptos de clases y objetos. Este tipo de programación se utiliza para estructurar el

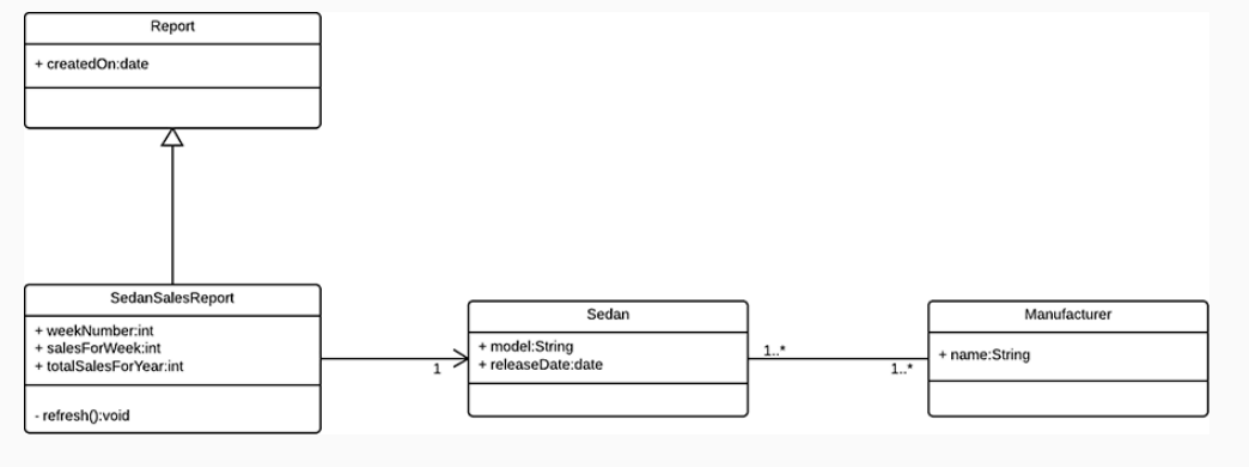
software como fragmentos simples y reutilizables de planes de código (clases) para crear instancias individuales de objetos.

3.1.2. Diagramas de UML

Mediante el señor (Mancuzo, 2021) nos dice que el lenguaje de modelado unificado (UML) se creó para proporcionar un lenguaje de modelado visual general, semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de la estructura y el comportamiento de los sistemas de software complejos. UML tiene aplicaciones más allá del desarrollo de software, p. Por ejemplo, en un flujo de proceso en la industria manufacturera.

- **Clases :** Una clase es el elemento principal de un diagrama y, como sugiere su nombre, una clase representa una clase en un paradigma orientado a objetos. Estos tipos de elementos se utilizan a menudo para representar conceptos o entidades "comerciales". Una clase define un conjunto de objetos que comparten propiedades, condiciones y significado comunes.
- **Casos de Uso:** (Cabrera, 2022) nos dice que un diagrama de casos de uso le permite visualizar las posibles interacciones que un usuario o cliente podría tener con el sistema. Sin embargo, los diagramas de casos de uso, utilizados anteriormente en la programación de computadoras, se han vuelto populares en las industrias minorista y de servicio al cliente para explicar las interacciones del cliente con una empresa o negocio.

Imagen 1. Diagrama UML



Los diagramas uml muestran la composición estática del sistema y nos dará a conocer las relaciones de clases que puede tener por medio del sr (Mancuzo, 2021)

3.1.3. Encapsulamiento

(Lara, 2015) nos da a conocer que el proceso de almacenar en una misma parte los elementos de una abstracción que conforman su estructura y comportamiento; se utiliza para separar la interfaz contractual de una abstracción de su implementación. Existen tres niveles de acceso para el encapsulamiento, los cuales son:

- público(public)
- protegido(protected)
- privado(private)

3.1.4. Constructores

Un constructor es un elemento de clase cuyo identificador corresponde a la clase en cuestión y cuyo propósito es implementar y controlar cómo se inicializan las instancias de una clase dada, ya que Java no permite que las variables miembro de nuevas instancias permanezcan inicializadas.

Imagen 2. Constructores de Kelvin Quezada

```
public Administrativos(String dni, String apellidos, String nombres,
   String sexo, int edad,String cargo ) {
   this.dni = dni;
   this.apellidos = apellidos;
   this.nombres = nombres;
   this.sexo = sexo;
   this.edad = edad;
   this.cargo=cargo;
}
```

El constructor es el que identifica cada atributo de la clase para realizar una instancia de una clase dada.

3.1.5. Métodos

Un método Java es una pieza de código que realiza una tarea relacionada con un objeto, un método es básicamente una función que pertenece a un objeto o una clase.

Set: Un método Java es una pieza de código que realiza una tarea relacionada con un objeto, un método es básicamente una función que pertenece a un objeto o una clase.

Get: El método get es un método público al igual que una colección, pero el método get se encarga de mostrar la propiedad del objeto o el valor de la propiedad encapsulado en la clase correspondiente, es decir, declarado o protegido por la palabra reservada private.

Imagen 3. Metodos Get y Set por Kelvin Quezada

```
//generamos los metodos get nos devuelve el valor de dni
public String getDni() {
   return dni;
}
//generamos los métodos set no devuelve nada solo establece datos
public void setDni(String dni) {
   this.dni = dni;
}
```

Es muy importante para los métodos get y set ya que son muy importante que nos devuelve un valor del método, y también establecen datos.

3.1.6. Código Limpio

Según (Paredes, 2022) el código limpio no es un conjunto rígido de reglas, sino un conjunto de principios que ayudan a crear un código intuitivo y fácilmente modificable. Intuitivo en este caso significa que cualquier desarrollador profesional puede entenderlo de inmediato. El código fácilmente personalizable tiene las siguientes características:

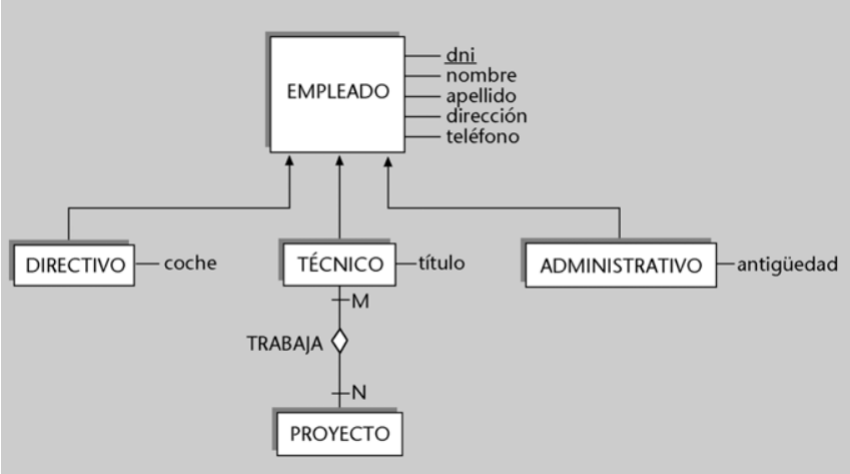
- La secuencia de ejecución de todo el programa es lógica y la estructura es simple.
- La relación entre las diferentes partes del código es claramente visible.
- La tarea o función de cada clase, función, método y variable se puede entender de un vistazo.

3.2. Generalización/ Especialización

Según (Vazquez) la relación de especialización/generalización (o de herencia) entre dos clases. Esta relación se considera propia de los lenguajes de POO. Una relación de herencia de la clase B (subclase, o clase hija) con respecto a (superclase o clase base) nos permite decir que la clase B obtiene todos los métodos y atributos de la clase A, y que luego puede añadir algunas características propias.

En el caso anterior se supone que utilizaremos la relación de herencia para decir que la clase B hereda de la clase A. Por medio de este mecanismo, la clase B comparte todas las características (atributos o estado y métodos o comportamiento) de la clase A. Esto no impide que se le pueda añadir a la clase B características adicionales (de nuevo, tanto atributos como métodos), o incluso, se modifique el comportamiento (la definición, no la declaración) de alguno de los métodos heredados.

Imagen 4. Generalización/especificación



En la generalización/especialización, es las características de los atributos o interrelaciones de la entidad superclase se propagan hacia las entidades subclase ya que denomina herencia de propiedades según (Dataprix, 2009)

La Herencia

Las instancias de una clase incluyen también las de su superclase o superclases. Como resultado, además de los atributos y métodos específicos de la clase, también utilizan los definidos en la superclase.

Esta capacidad se llama herencia, lo que significa que una clase hereda las propiedades y funciones de sus superclases para que sus instancias puedan usarlas. Al colocar el símbolo delante de los atributos y métodos heredados en las subclases, UML brinda la opción de representarlos.

3.2.1. Implementación

Según (Solís, 2015) la programación e implementación de clases trata de amoldarse al modo de pensar del hombre y ajustarse a su forma de analizar y gestionar los problemas que

tenga que desarrollar. Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común. La implementación de una clase comprende dos componentes: la declaración y el cuerpo de la clase.

Ejemplo de sintaxis

Definidos los tipos y funciones pasamos a la declaración e implementación de clases con ello podemos identificar la estructura de la programación

3.3. Gestión de Defecto testing

3.3.1. Verificacion y Validacion

Los procesos de verificación y análisis que aseguran que el software que se está desarrollando está en línea con su especificación y satisface las necesidades de los clientes se denominan verificación y validación. Un proceso de ciclo de vida completo es VandV. Las revisiones de los requisitos vienen primero, luego las revisiones de los diseños y el código, y finalmente las pruebas del producto. En cada etapa del proceso de desarrollo de software, se llevan a cabo actividades de V&V. A pesar de la facilidad con la que pueden confundirse, Boehm (1979) describió sucintamente la diferencia entre verificación y validación de la siguiente manera.

Verificación: las responsabilidades de Verificación incluyen asegurarse de que el software cumpla con sus especificaciones. Se confirma el cumplimiento del sistema con los requisitos funcionales y no funcionales establecidos.

La validación es un proceso que se usa más ampliamente: El software necesita ser revisado para ver si cumple con las expectativas del cliente. El software se prueba para ver si

funciona como el usuario espera en lugar de lo especificado, lo que va más allá de determinar si el sistema cumple con sus especificaciones.

Primero, es crucial validar los requisitos del sistema. Es sencillo cometer errores y omisiones durante la fase de análisis de requisitos del sistema y, en tales casos, el software terminado no estará a la altura de las expectativas del cliente. Sin embargo, en realidad, no todos los problemas que presenta una aplicación no se pueden encontrar a través de la validación de requisitos. Cuando el sistema se haya implementado por completo, es posible que se encuentren algunos errores en los requisitos. (Drake, 2009)

3.3.2. Polimorfismo

Según (Fernandez, s.f.) al crear objetos con comportamientos compartidos, el polimorfismo nos permite procesar objetos de diversas formas. Implica tener la capacidad de mostrar la misma interfaz de usuario para varios formularios o tipos de datos subyacentes. Los objetos pueden reemplazar comportamientos primarios comunes con comportamientos secundarios particulares mediante el uso de la herencia. La sobrecarga de métodos y la anulación de métodos son dos formas en que el polimorfismo permite que el mismo método lleve a cabo varios comportamientos.

Las clases con tipos compatibles se pueden usar en cualquier parte de nuestro código gracias al polimorfismo. La compatibilidad de tipos en Java se refiere a cómo una clase se extiende a otra o cómo una clase implementa una interfaz. Informalmente: Podemos enlazar referencias de sus hijas a una referencia de tipo padre. Cualquier instancia de una clase que implemente la interfaz se puede conectar a una referencia de tipo de interfaz.

Imagen 5. Polimorfismo por Kelvin Quezada

```
public class Animal {
    String nombre;
    private int edad;
    private String tipo;
    public Animal(String nombre, int edad, String tipo) {...5 lines }

    public void comer() {
        System.out.println("El " + tipo + " " + nombre + " está comiendo.");
    }
    public void dormir() {
        System.out.println("El " + tipo + " " + nombre + " está durmiendo.");
    }
}
```

El polimorfismo es una poderosa característica de la POO que permite a las clases compartir una interfaz común y, al mismo tiempo, proporcionar implementaciones específicas según sus necesidades individuales según Kelvin Quezada

3.4. Modelo Vista controlador

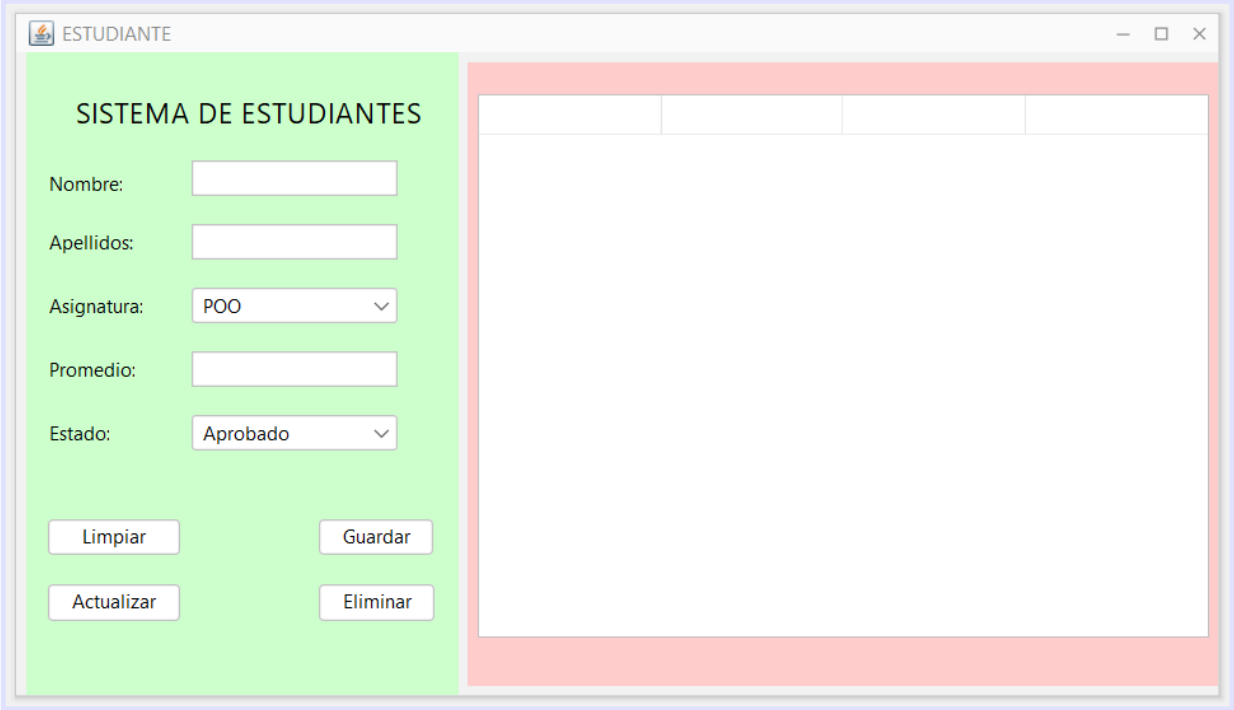
3.4.1. El Modelo

El señor (Pantoja, 2004) el modelo es un conjunto de clases que representa la información del mundo real que debe procesar el sistema. Sin tener en cuenta los métodos por los cuales se generarán los datos o cómo se mostrarán, es decir, sin tener una conexión con otra entidad dentro de la aplicación. Las vistas y un controlador están ocultos para el modelo, que no los conoce. SmallTalk sugiere que el modelo en realidad consta de dos submódulos: el modelo de dominio y el modelo de red. Si bien ese enfoque suena intrigante, no es práctico porque debe haber interfaces que permitan que los módulos se comuniquen entre sí. solicitud.

3.4.2. La Vista

Las vistas se encargan de mostrar al usuario los datos del modelo. Las vistas y el modelo tienen una relación de muchos a uno, lo que significa que cada vista tiene un modelo asociado, aunque puede haber varias vistas vinculadas a un solo modelo. De esta forma, por ejemplo, podría tener una vista que muestre la hora del sistema como un reloj analógico y otra vista que muestre la misma información como un reloj digital. La vista solo requiere los datos necesarios del modelo para ejecutar una visualización. La vista se altera a través de notificaciones que produce el modelo de aplicación cada vez que se realiza una acción que implica un cambio en el modelo de dominio. En pocas palabras, cuando se produce una modificación, la representación visual del modelo vuelve a dibujar los componentes necesarios.

Imagen 6. Vista Por Kelvin Quezada



Según Kelvin Quezada la modelo vista son los datos que visualizamos en la ventana ya que interactúa el usuario.

3.4.3. El Controlador

Mediante el señor (Hernandez, 2021) la responsabilidad del controlador es extraer, modificar y proporcionar datos al usuario. Esencialmente, el controlador es el enlace entre y el modelo a través de las funciones getter y setter, el controlador extrae datos del modelo e inicializa las vistas si hay alguna actualización desde las vistas, modifica los datos con una función setter.

Imagen 7. Controlador Por Kelvin Quezada

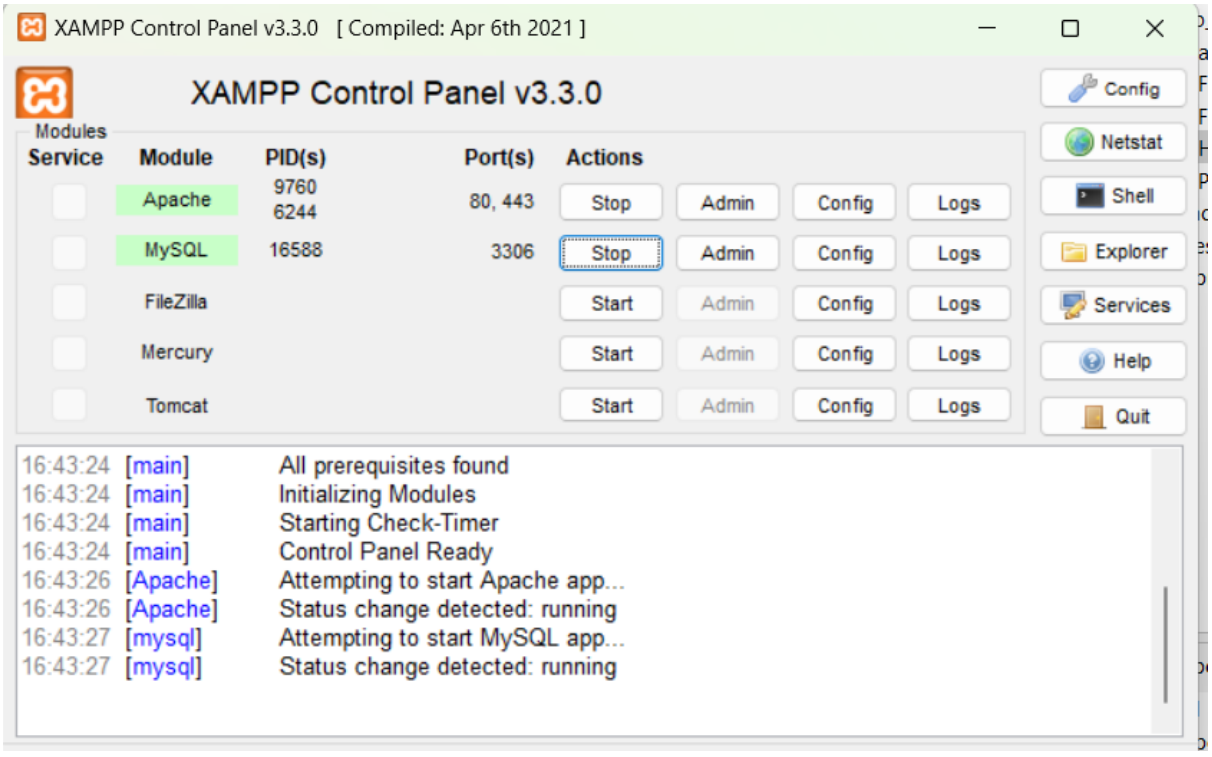
```java
public class Conexion {
    private static final String  driver="com.mysql.jdbc.Driver";
    private static final String  user="root";
    private static final String  pass="";
    private static final String  url="jdbc:mysql://localhost:3306/registroUsuario";

    Connection conectar=null;

    public Connection conexion(){
        try{
            Class.forName(className:driver);
            conectar=(Connection) DriverManager.getConnection(url,user,password: pass);
        }catch (Exception e){
            JOptionPane.showMessageDialog(parentComponent: null, "error de conexion"+e.getMessage());
        }
        return conectar;
    }

}
```

Según Kelvin Quezada el controlador es el intermedio del modelo y la vista ya que recibe las interacciones entre el usuario y actualiza el modelo en consecuencia.

3.4.4. Que es Xampp

Según (Jesús, 2022) XAMPP proporciona una forma sencilla y rápida de configurar un entorno de desarrollo web local en una computadora personal, lo que facilita la creación y prueba de aplicaciones antes de ser desplegadas en un servidor en línea. Esto lo hace especialmente útil para desarrolladores web, ya que pueden trabajar en sus proyectos sin necesidad de una conexión a Internet y tener un entorno aislado para realizar pruebas y experimentar sin afectar el servidor en producción.

Imagen 8. Xampp



Según (Jesús, 2022) nos dice que es un software libre que ayuda mucho para los programadores ya que se trabaja para proyectos, paneles controles con base de datos.

3.5. Desarrollo

3.5.1. Planteamiento del Problema

Realizar un software que ayude al registro de los profesores conocer sus datos personales y si se encuentra activado en la zona laboral tendremos una base de datos de todos los profesores que están activo o ya no está activo. En la otra opción será de estudiantes donde el estudiante se registrara escogerá la asignatura y sabrá si aprueba o reprueba la materia. Mediante en la zona de horario nos dará a conocer el día, la hora y la materia que está entrando cada materia.

3.5.2. Pasos para crear Una base de datos

Primeramente abrimos Xampp y seleccionamos apache y mysql y nos dará a conocer los puestos de nuestro ports para ingresar a nuestro http://localhost/phpmyadmin/

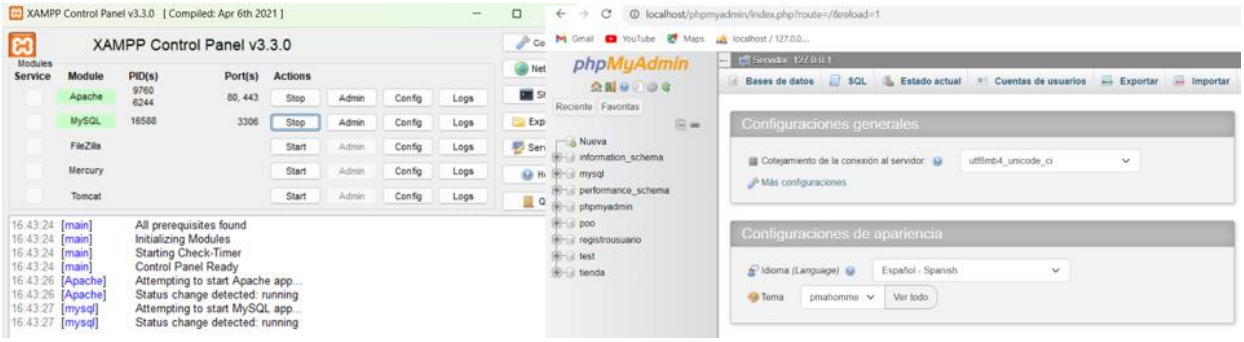


Imagen 9. Conexión de XAMPP Kelvin Quezada

Segundo paso nos dirigimos a nueva nos dirá que ingresemos un nombre de nuestro proyecto para la base de datos en este caso se creo una base de datos llamado RegistroUsuario y colocamos en crear después de ello creamos adentro de nuestro proyecto otra base de datos llamados estudiante, profesor y estudiante.

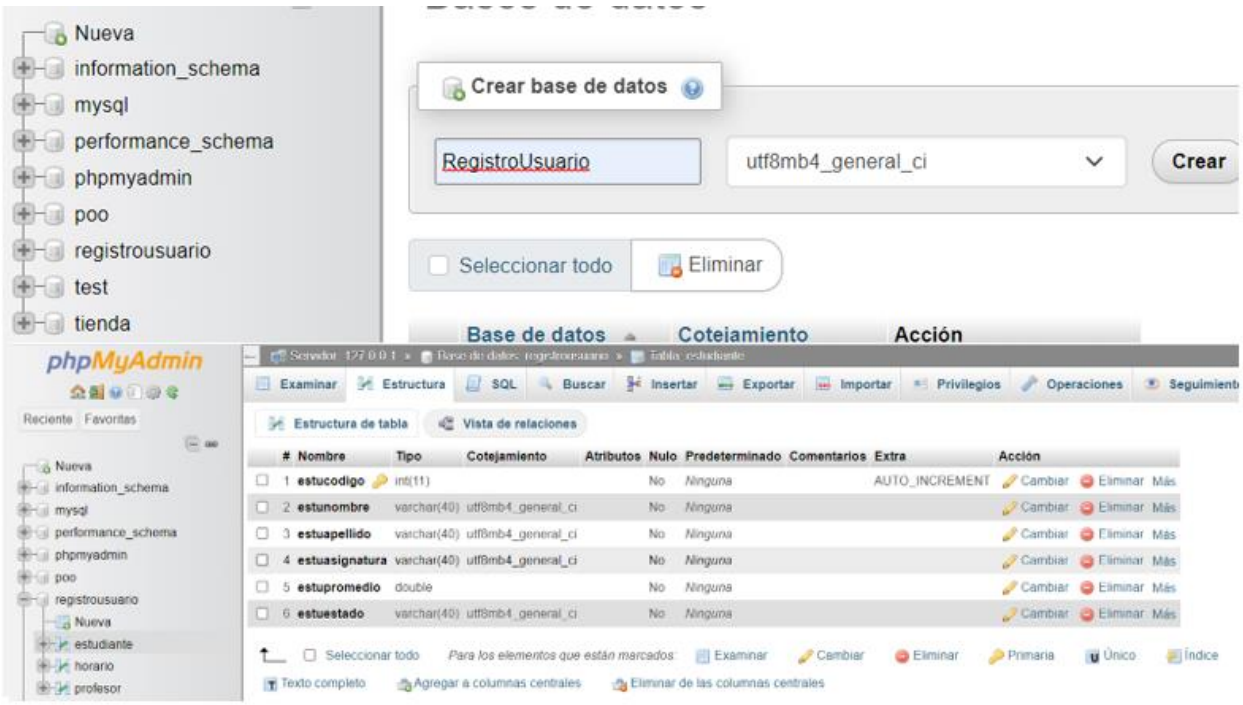


Imagen 10. phpMyAdmi por Kelvin Quezada

3.5.3. Pasos para crear las interfaces

Una vez que hayamos creado nuestra base de datos creamos nuestras clases que son Principal, Form_estudiante, Form_porfesor, Horario.

En esta ocasión ceamos nuestra clase Principal utilizamos un Menu Bar y utilizamos 3 Menu Item, en la cual vamos a colocar opciones que es Estudiante, Profesor, Horario, después damos clic en cada opción y configuramos nuestro métodos para que muestre nuestras clases secundarias.

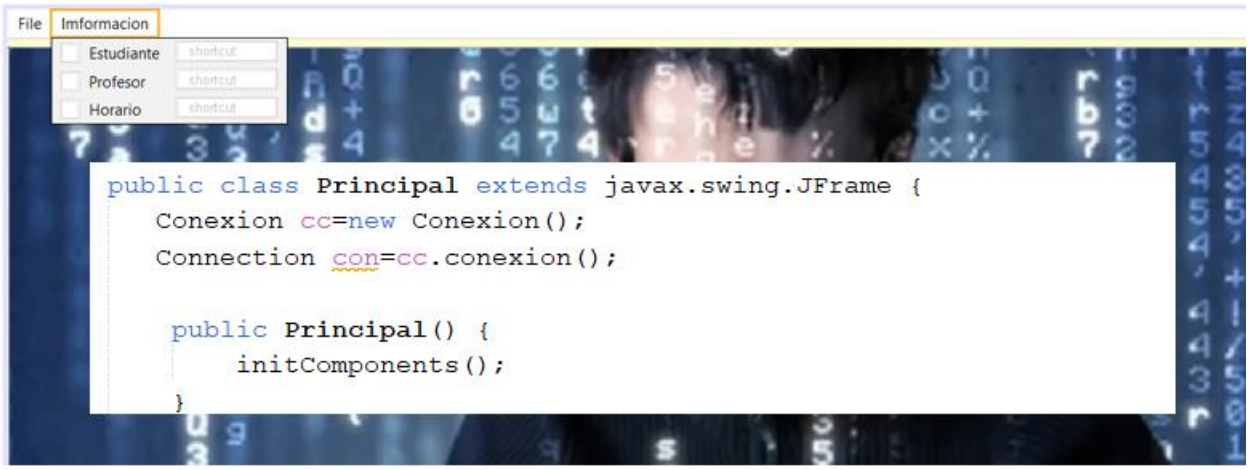


Imagen 11. Menu Principal por Kelvin Quezada

Creamos nuestros métodos en para llamar a nuestras interfaces secundarias ya que llamaremos a nuestras clase y creamos una objeto para así instancias nuestra opciones.

```
private void opcEstudianteActionPerformed(java.awt.event.ActionEvent evt) {
    Form_estudiante form1=new Form_estudiante();
    Escritorio.add(comp: form1);
    form1.toFront();
    form1.setVisible(aFlag: true);
}

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    Form_profe form2=new Form_profe();
    Escritorio.add(comp: form2);
    form2.toFront();
    form2.setVisible(aFlag: true);
}

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    Horario form3=new Horario();
    Escritorio.add(comp: form3);
    form3.toFront();
    form3.setVisible(aFlag: true);
}
```

Imagen 12. Configuraciones de nuestras selección por Kelvin Quezada

Creamos una clase llamada Conexión para llamar a nuestra base de datos mysql para poder acceder al localhost de nuestra base de datos.

```
public class Conexion {
    private static final String  driver="com.mysql.jdbc.Driver";
    private static final String  user="root";
    private static final String  pass="";
    private static final String  url="jdbc:mysql://localhost:3306/registroUsuario";

    Connection conectar=null;

    public Connection conexion(){
        try{
            Class.forName(className:driver);
            conectar=(Connection) DriverManager.getConnection(url,user,password: pass);
        }catch (Exception e){
            JOptionPane.showMessageDialog(parentComponent: null, "error de conexion"+e.getMessage());
        }
        return conectar;
    }

}
```

Imagen 13. Conexion de localhost por Kelvin Quezada

3.5.4. Clase Estudiante

Es este caso tenemos nuestra primera clase llamada Form_estudiante donde vamos utilizar 6 label para ingresar notas de información, también utilizamos 3 Text Field, dos combo box, 4 botones y una caja de Table.

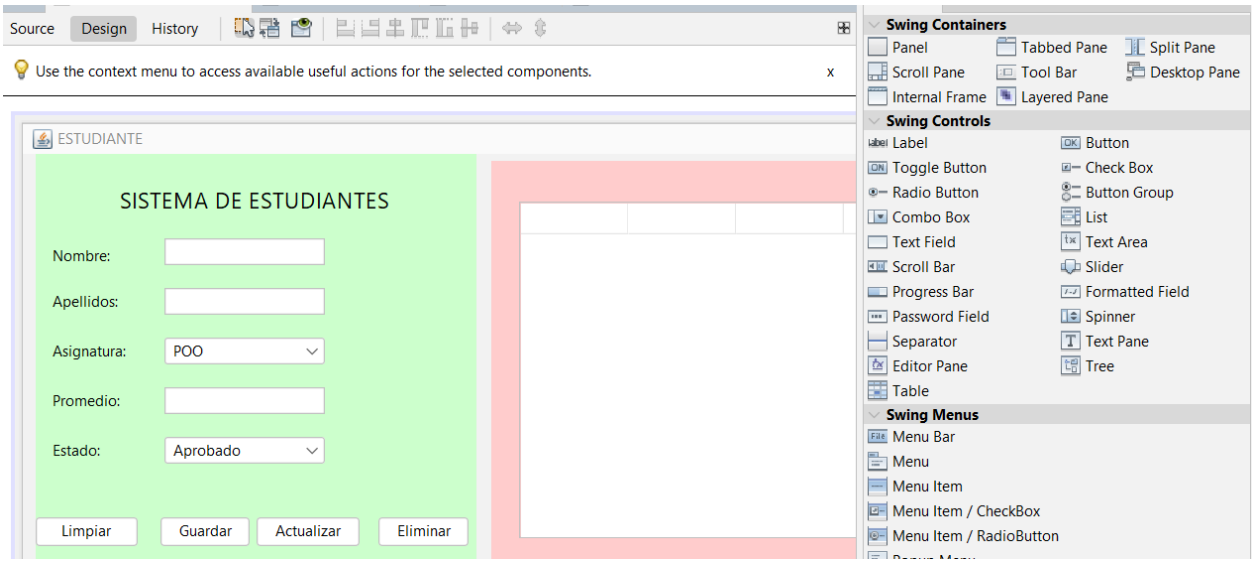


Imagen 14. Estudiante por Kelvin Quezada

Creamos nuestro método limpiarDatos para borrar la información de nuestra caja de texto por medio del botón limpiar.

Imagen 15. limpiar Estudiante por Kelvin Quezada

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt)
    limpiarDatos();
}

  public void limpiarDatos(){
   txtNombre.setText(t: "");
   txtApellidos.setText(t: "");
   cboAsignatura.setSelectedItem(anObject: null);
   txtPromedio.setText(t: "");
   cboEstado.setSelectedItem(anObject: null);
}
```

Borrara toda las caja de texto y los combo box por kelvin quezada

Imagen 16.guardarDatos de clase Estudiante por Kelvin Quezada

```
public void guardarDatos(){
try {
    String apellido = txtApellidos.getText().trim();
    String nombre = txtNombre.getText().trim();
    String promedioStr = txtPromedio.getText().trim();
    if (apellido.isEmpty() || nombre.isEmpty() || promedioStr.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de guardar");
        return;
    }
    double promedio = Double.parseDouble(s: promedioStr);
    String asignatura = cboAsignatura.getSelectedItem().toString();
    String estado = cboEstado.getSelectedItem().toString();
    String SQL = "INSERT INTO estudiante (estu_apellido, estu_nombre, estu_asignatura, estu_promedio, estu_estado) VALUES (?, ?, ?, ?, ?)";
    PreparedStatement pst=(PreparedStatement) con.prepareStatement(string:SQL);
    pst.setString(parameterIndex: 1, x: txtNombre.getText());
    pst.setString(parameterIndex: 2, x: txtApellidos.getText());
    int seleccion=cboAsignatura.getSelectedIndex();
    pst.setString(parameterIndex: 3,x: cboAsignatura.getItemAt(index: seleccion));
    pst.setString(parameterIndex: 4,x: txtPromedio.getText());
    int seleccion2=cboEstado.getSelectedIndex();
    pst.setString(parameterIndex: 5,x: cboEstado.getItemAt(index: seleccion2));
    pst.execute();
     JOptionPane.showMessageDialog(parentComponent: null, message: "Registro exitoso");
    limpiarDatos();
    // Mostramos nuevamente los datos actualizados en la tabla
    mostrarDatos();
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
} catch (Exception e) {
    JOptionPane.showMessageDialog(parentComponent: null, "Error al guardar el registro: " + e.getMessage());
}
```

Después creamos un método guardarDatos donde llamaremos a nuestra caja de texto y a nuestras combo box para seleccionar las la asignatura y el estado por Kelvin Quezada.

Imagen 17. MostrarDatos de Estudiante por Kelvin Quezada

```
//creamos un metodo para mostrar datos
  public void mostrarDatos(){
      //utilizamos un titulo para llamar a nuestro arreglo en una tabla
      String titulos[]={"Codigo","Apellido","Nombre","Asignatura","Promedio","Estado"};
      //colocamos la dimencion del arreglo que es [6];
      String registro[]=new String[6];
      DefaultTableModel modelo=new DefaultTableModel(data: null,columnNames: titulos);
      String SQL="SELECT * FROM `estudiante`";
      try{
          Statement st=(Statement) con.createStatement();
          ResultSet rs=st.executeQuery(string:SQL);
          while(rs.next()){
              //llamamos a nuestros atributos que tenemos en nuestra base de datos de localhost
              registro[0]=rs.getString(string:"estu_codigo");
              registro[1]=rs.getString(string:"estu_nombre");
              registro[2]=rs.getString(string:"estu_apellido");
              registro[3]=rs.getString(string:"estu_asignatura");
              registro[4]=rs.getString(string:"estu_promedio");
              registro[5]=rs.getString(string:"estu_estado");
              modelo.addRow(rowData:registro);
          }
          //llamamos a nuestra tabla que seba a registar todo los ingresos
          tablaEstudiante.setModel(dataModel:modelo);
      }catch( Exception e){
          //si la informacion es incorrecta saldra que hay un error de los datos
          JOptionPane.showMessageDialog(parentComponent: null,"error mostrar datos"+e.getMessage());
      }
  }
```

Realizamos un método con arreglos creamos un nuevo objeto para y colocamos el limite de nuestros arreglos y llamamos a nuestros atributos de como tenemos a guardados nuestro registro en la base de datos por Kelvin Quezada.

Imagen 18. ActualizarDatos de la clase Estudiante por Kelvin Quezada

```
public void actualizarDatos(){
    try {
    int filaSeleccionada = tablaEstudiante.getSelectedRow();
    if (filaSeleccionada == -1) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Selecciona un registro para actualizar");
        return;
    }
    String estu_codigo = (String) tablaEstudiante.getValueAt(row:filaSeleccionada, column:0);
    String apellido = txtApellidos.getText().trim();
    String nombre = txtNombre.getText().trim();
    String promedioStr = txtPromedio.getText().trim();
    if (apellido.isEmpty() || nombre.isEmpty() || promedioStr.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de actualizar");
        return;
    }
    double promedio = Double.parseDouble(s: promedioStr);
    String asignatura = cboAsignatura.getSelectedItem().toString();
    String estado = cboEstado.getSelectedItem().toString();
    String SQL = "UPDATE estudiante SET estu_apellido=?, estu_nombre=?, estu_asignatura=?, estu_promedio=?, estu_estado=? WHERE estu_codigo=

    String dao=(String)tablaEstudiante.getValueAt(row:filaSeleccionada,column:0);
    PreparedStatement pst=(PreparedStatement) con.prepareStatement(string:SQL);
    pst.setString(parameterIndex: 1, x: apellido);
    pst.setString(parameterIndex: 2, x: nombre);
    pst.setString(parameterIndex: 3, x: asignatura);
    pst.setDouble(parameterIndex: 4, x: promedio);
    pst.setString(parameterIndex: 5, x: estado);
    pst.setString(parameterIndex: 6, x: estu_codigo);
    pst.executeUpdate();
    JOptionPane.showMessageDialog(parentComponent: null, message: "Se ha actualizado correctamente el registro");
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
} catch (Exception e) {
    JOptionPane.showMessageDialog(parentComponent: null, "Error en la actualización: " + e.getMessage());
}
```

Creamos un método llamamos actualizar tabla ya que el usuario se equiboco por error algún nombre o algún numero lo puede modificar por medio de la tablaEstudiante por medio del SQL llamaremos a nuestros atributos que tenemos en la base de datos.

3.5.5. Clase Profesor

Es este caso tenemos nuestra primera clase llamada Form_Profesor donde vamos utilizar 6 label para ingresar notas de información, también utilizamos 4 Text Field, 2 combo box, 4 botones y una caja de Table.

Imagen 19. Profesor por Kelvin Quezada

Imagen 20. mostrarDatos del la clase Profesor

```
//creamos un metodo para mostrar datos
  public void mostrarDatos(){
      //utilizamos un titulo para llamar a nuestro arreglo en una tabla
      String titulos[]={"Codigo","Apellido","Nombre","Asignatura","Promedio","Estado"};
      //colocamos la dimencion del arreglo que es [6];
      String registro[]=new String[6];
      DefaultTableModel modelo=new DefaultTableModel(data: null,columnNames: titulos);
      String SQL="SELECT * FROM `estudiante`";
      try{
          Statement st=(Statement) con.createStatement();
          ResultSet rs=st.executeQuery(string:SQL);
          while(rs.next()){
              //llamamos a nuestros atributos que tenemos en nuestra base de datos de localhost
              registro[0]=rs.getString(string:"estu_codigo");
              registro[1]=rs.getString(string:"estu_nombre");
              registro[2]=rs.getString(string:"estu_apellido");
              registro[3]=rs.getString(string:"estu_asignatura");
              registro[4]=rs.getString(string:"estu_promedio");
              registro[5]=rs.getString(string:"estu_estado");
              modelo.addRow(rowData: registro);
          }
          //llamamos a nuestra tabla que seba a registar todo los ingresos
          tablaEstudiante.setModel(dataModel:modelo);
      }catch( Exception e){
          //si la informacion es incorrecta saldra que hay un error de los datos
          JOptionPane.showMessageDialog(parentComponent: null,"error mostrar datos"+e.getMessage());
      }
  }
```

Tendremos nuestras interfaces de profesor con botones y caja de datos que se aguardara los registros.

Imagen 21. limpiar Profesor por Kelvin Quezada

```
    private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
            limpiarDatos();

    }
public void limpiarDatos(){
        txtNombres.setText(t: "");
        txtApellidos.setText(t: "");
        txtCelular.setText(t: "");
        txtmail.setText(t: "");
        bnoTitulo.setSelectedItem(anObject: null);
        bnoEstados.setSelectedItem(anObject: null);

    }
```

Borrara todas las cajas de texto y el combo box por kelvin quezada

Imagen 22. MostrarDatos del Profesor por Kelvin Quezada

```
    public void mostrarDatos(){

        String titulos[]={"Nombre","Apellido","Celular","mail","Titulo","Estado"};
        String registro[]=new String[6];
        DefaultTableModel modelo=new DefaultTableModel(data: null,columnNames: titulos);
        String SQL="SELECT * FROM `profesor`";
        try{
            Statement st=(Statement) con.createStatement();
            ResultSet rs=st.executeQuery(string:SQL);
            while(rs.next()){

                registro[0]=rs.getString(string:"pro_nombre");
                registro[1]=rs.getString(string:"pro_apellido");
                registro[2]=rs.getString(string:"pro_celular");
                registro[3]=rs.getString(string:"pro_mail");
                registro[4]=rs.getString(string:"pro_titulo");
                registro[5]=rs.getString(string:"pro_estado");
                modelo.addRow(rowData: registro);
            }
            tablaProfesor.setModel(dataModel:modelo);
        }catch( Exception e){
            JOptionPane.showMessageDialog(parentComponent: null,"error mostrar datos"+e.getMessage());
        }
    }
```

Realizamos un método con arreglos creamos un nuevo objeto para y colocamos el limite de nuestros arreglos y llamamos a nuestros atributos de como tenemos a guardados nuestro registro en la base de datos por Kelvin Quezada.

Imagen 23. actualizarDatos de la clase Profesor por Kelvin Quezada

```
public void actualizasDatos(){
    try {
         int filaSeleccionada = tablaProfesor.getSelectedRow();
         if (filaSeleccionada == -1) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Selecciona un registro para actualizar");
            return;
        }
        String nombre = txtNombres.getText().trim();
        String apellido = txtApellidos.getText().trim();
        String celular = txtCelular.getText().trim();
        String mail = txtmail.getText().trim();
        if (nombre.isEmpty() || apellido.isEmpty() || celular.isEmpty() || mail.isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de actualizar");
            return;
        }
        String titulo = bnoTitulo.getSelectedItem().toString();
        String estado = bnoEstados.getSelectedItem().toString();
        String SQL = "UPDATE profesor SET pro_nombre=?, pro_apellido=?, pro_celular=?, pro_mail=?, pro_titulo=?, pro_estado=? ";
        String dao=(String)tablaProfesor.getValueAt(row:filaSeleccionada,column:0);
        PreparedStatement pst=(PreparedStatement) con.prepareStatement(string:SQL);
        pst.setString(parameterIndex: 1, x: nombre);
        pst.setString(parameterIndex: 2, x: apellido);
        pst.setString(parameterIndex: 3, x: celular);
        pst.setString(parameterIndex: 4, x: mail);
        pst.setString(parameterIndex: 5, x: titulo);
        pst.setString(parameterIndex: 6, x: estado);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(parentComponent: null, message: "Se ha actualizado correctamente el registro");
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(parentComponent: null, "Error en la actualización: " + e.getMessage());
    }
}
```

Creamos un método llamamos actualizar tabla ya que el usuario se equivocó por error algún nombre o algún numero lo puede modificar por medio de la tablaProfesor por medio del SQL llamaremos a nuestros atributos que tenemos en la base de datos.

Imagen 24. eliminar de la clase Profesor por Kelvin Quezada

```
    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    eliminarDatos();
        // TODO add your handling code here:
    }
public void eliminarDatos(){
        int filaSeleccionada=tablaProfesor.getSelectedRow();
        try {
            String estu_nombre = (String) tablaProfesor.getValueAt(row:filaSeleccionada, column:0);
            String SQL="DELETE FROM profesor "+tablaProfesor.getValueAt(row:filaSeleccionada, column:0);
            Statement st=(Statement) con.createStatement();
            int n=st.executeUpdate(string:SQL);
            if(n>=0){
                JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Eliminado correctamente");
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent: null,"Error en eliminar registro"+e.getMessage());
        }
    }
```

El usuario podrá eliminar su registro en la base datos del sistema.

Imagen 25. guardarDatos de la clase Profesor por Kelvin Quezada

```
public void guardarDatos(){
    try {
        String nombre = txtNombres.getText().trim();
        String apellido = txtApellidos.getText().trim();
        String celular = txtCelular.getText().trim();
        String mail = txtmail.getText().trim();
        if (nombre.isEmpty() || apellido.isEmpty() || celular.isEmpty() || mail.isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de actualizar");
            return;
        }
        String titulo = bnoTitulo.getSelectedItem().toString();
        String estado = bnoEstados.getSelectedItem().toString();
        String SQL = "INSERT INTO profesor (pro_nombre, pro_apellido, pro_celular, pro_mail, pro_titulo, pro_estado) VALUES (?, ?, ?, ?, ?, ?)";
        PreparedStatement pst=(PreparedStatement) con.prepareStatement(string:SQL);
        pst.setString(parameterIndex: 1, x: txtNombres.getText());
        pst.setString(parameterIndex: 2, x: txtApellidos.getText());
        pst.setString(parameterIndex: 3, x: txtCelular.getText());
        pst.setString(parameterIndex: 4,x: txtmail.getText());
        int seleccion1=bnoTitulo.getSelectedIndex();
        pst.setString(parameterIndex: 5,x: bnoTitulo.getItemAt(index: seleccion1));
        int seleccion2=bnoEstados.getSelectedIndex();
        pst.setString(parameterIndex: 6,x: bnoEstados.getItemAt(index: seleccion2));
         //aqui ejecutamos la consulta
        pst.execute();
         JOptionPane.showMessageDialog(parentComponent: null, message: "Registro exitoso");
        // Limpiamos los campos después de guardar
        limpiarDatos();
        // Mostramos nuevamente los datos actualizados en la tabla
        mostrarDatos();
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(parentComponent: null, "Error al guardar el registro: " + e.getMessage());
    }
}
```

Se guardará los datos que fueron registra en la base de datos sql

3.5.6. Clase Horario

Es este caso tenemos nuestra primera clase llamada Horario donde vamos utilizar 4 label para ingresar notas de información, también utilizamos 2 Text Field, 2 combo box, 4 botones y una caja de Table.

Imagen 26. Horario Por Kelvin Quezada

Tendremos nuestras interfaces de profesor con botones y caja de datos que se aguardara los registros.

Imagen 27. limpiar Horario por Kelvin Quezada

```
    private void bntLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
limpiarDatos();
        // TODO add your handling code here:
    }

    public void limpiarDatos(){
        txtDia.setText(t: "");
        txtHora.setText(t: "");
        bnoEstado.setSelectedItem(anObject: null);

    }
```

Borrara todas las cajas de texto y el combo box por kelvin quezada

Imagen 28. MostrarDatos de Horario por Kelvin Quezada

```
    public void mostrarDatos(){
    //realizo un arreglo
    String titulos[]={"DIA","HORA","MATERIA","ESTADO"};
    String registro[]=new String[4];
    //instanciamos
    DefaultTableModel modelo=new DefaultTableModel(data: null,columnNames: titulos);
    String SQL="SELECT * FROM `horario`";
    try{
        Statement st=(Statement) con.createStatement();
        ResultSet rs=st.executeQuery(string:SQL);
        while(rs.next()){
            //llamamos miestos atributos que tenemos en nuestra base de datos de localhost
            registro[0]=rs.getString(string:"ho_dia");
            registro[1]=rs.getString(string:"ho_hora");
            registro[2]=rs.getString(string:"ho_materia");
            registro[3]=rs.getString(string:"ho_estado");
            modelo.addRow(rowData: registro);
        }
        //llamo a la tablla de los
        tablaHorario.setModel(dataModel:modelo);
    }catch( Exception e){
        JOptionPane.showMessageDialog(parentComponent: null,"error mostrar datos"+e.getMessage());
    }
}
```

Realizamos un método con arreglos creamos un nuevo objeto para y colocamos el limite de nuestros arreglos y llamamos a nuestros atributos de como tenemos a guardados nuestro registro en la base de datos por Kelvin Quezada.

Imagen 29. Se eliminarDatos en clase Horario por Kelvin Quezada

```
public void eliminarDatoS(){
    int filaSeleccionada=tablaHorario.getSelectedRow();
    try {
        String estu_codigo = (String) tablaHorario.getValueAt(row:filaSeleccionada, column:0);
        String SQL="DELETE FROM estudiante where estu_codigo="+tablaHorario.getValueAt(row:filaSeleccionada, column:0);
        Statement st=(Statement) con.createStatement();
        int n=st.executeUpdate(string:SQL);
        if(n>=0){
            JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Eliminado correctamente");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(parentComponent: null,"Error en eliminar registro"+e.getMessage());
    }
```

Se eliminara el registro que se aguardo de la base de datos siempre y cuándo el desea eliminar sus datos.

Imagen 30. GuardarDatos de la clase Horario por el Kelvin Quezada

```
private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
guardarDatos();
}

public void guardarDatos(){
try {
    String dia = txtDia.getText().trim();
    String hora = txtHora.getText().trim();
    if (dia.isEmpty() || hora.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de actualizar");
        return;
    }
    String materia = bnoMateria.getSelectedItem().toString();
    String estado = bnoEstado.getSelectedItem().toString();
    String SQL = "INSERT INTO horario (ho_dia, ho_hora,ho_materia, ho_estado) VALUES (?, ?, ?,?)";
    PreparedStatement pst=(PreparedStatement) con.prepareStatement(string:SQL);
    pst.setString(parameterIndex: 1, x: txtDia.getText());
    pst.setString(parameterIndex: 2, x: txtHora.getText());
    int seleccion1=bnoMateria.getSelectedIndex();
    pst.setString(parameterIndex: 3,x: bnoMateria.getItemAt(index: seleccion1));
    int seleccion2=bnoEstado.getSelectedIndex();
    pst.setString(parameterIndex: 4,x: bnoEstado.getItemAt(index: seleccion2));
     //aqui ejecutamos la consulta
    pst.execute();
     JOptionPane.showMessageDialog(parentComponent: null, message: "Registro exitoso");
    // Limpiamos los campos después de guardar
    limpiarDatos();
    // Mostramos nuevamente los datos actualizados en la tabla
    mostrarDatos();
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
} catch (Exception e) {
    JOptionPane.showMessageDialog(parentComponent: null, "Error al guardar el registro: " + e.getMessage());
}
```

Se guardará los datos que fueron registra en la base de datos sql

Imagen 31. ActualizarDatos en la clase Horarios por Kelvin Quezada

```
private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
    actualizarDatos();
}

public void actualizarDatos(){
    try {
     int filaSeleccionada = tablaHorario.getSelectedRow();
     if (filaSeleccionada == -1) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Selecciona un registro para actualizar");
        return;
    }
    String dia = txtDia.getText().trim();
    String hora = txtHora.getText().trim();
    if (dia.isEmpty() || hora.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Completa todos los campos antes de actualizar");
        return;
    }

    String materia = bnoMateria.getSelectedItem().toString();
    String estado = bnoEstado.getSelectedItem().toString();
    String SQL = "UPDATE horario SET ho_dia=?, ho_hora=?, ho_estado=? ";
    String dao=(String)tablaHorario.getValueAt(row: filaSeleccionada,column: 0);
    PreparedStatement pst=(PreparedStatement) con.prepareStatement(string: SQL);
    pst.setString(parameterIndex: 1, x: dia);
    pst.setString(parameterIndex: 2, x: hora);
    pst.setString(parameterIndex: 3, x: materia);
    pst.setString(parameterIndex: 4, x: estado);
    // Ejecutamos la consulta
    pst.executeUpdate();
    JOptionPane.showMessageDialog(parentComponent: null, message: "Se ha actualizado correctamente el registro");

} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent: null, message: "El valor del promedio debe ser numérico");
} catch (Exception e) {
    JOptionPane.showMessageDialog(parentComponent: null, "Error en la actualización: " + e.getMessage());
}
```

Creamos un método llamamos actualizar tabla ya que el usuario se equivoco por error algún nombre o algún numero lo puede modificar por medio de la tablaHorario por medio del SQL llamaremos a nuestros atributos que tenemos en la base de datos.

3.6. Código del Programa

3.6.1. Código de Conexión

package controller;

import com.mysql.jdbc.Connection;

import java.sql.DriverManager;

import javax.swing.JOptionPane;

```
public class Conexion {
    private static final String  driver="com.mysql.jdbc.Driver";
    private static final String  user="root";
    private static final String  pass="";
    private static final String  url="jdbc:mysql://localhost:3306/registrousuario";

    Connection conectar=null;

    public Connection conexion(){
        try{
            Class.forName(driver);
            conectar=(Connection) DriverManager.getConnection(url,user,pass);
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, "error de conexion"+e.getMessage());
        }
        return conectar;
    }

}
```

3.6.2. Código Principal

```
import controller.Conexion;
import java.sql.Connection;
import javax.swing.JOptionPane;
```

```
public class Principal extends javax.swing.JFrame {
   Conexion cc=new Conexion();
   Connection con=cc.conexion();

    public Principal() {
        initComponents();
    }

    private void opcEstudianteActionPerformed(java.awt.event.ActionEvent evt) {
        Form_estudiante form1=new Form_estudiante();
        Escritorio.add(form1);
        form1.toFront();
        form1.setVisible(true);
    }

    private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        Form_profe form2=new Form_profe();
        Escritorio.add(form2);
        form2.toFront();
        form2.setVisible(true);
    }
```

```
    private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
        Form_horario form3=new Form_horario();
        Escritorio.add(form3);
        form3.toFront();
        form3.setVisible(true);
    }
```

3.6.3. Código For_estudiante

```
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;
import controller.Conexion;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import java.sql.ResultSet;

public class Form_estudiante extends javax.swing.JInternalFrame {
    Conexion cc=new Conexion();
    Connection con=cc.conexion();

    public Form_estudiante() {
        initComponents();
        mostrarDatos();
```

```
    }
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        limpiarDatos();
    }

      public void limpiarDatos(){
        txtNombre.setText("");
        txtApellidos.setText("");
        cboAsignatura.setSelectedItem(null);
        txtPromedio.setText("");
        cboEstado.setSelectedItem(null);
    }
    private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
        guardarDatos();
    }

    public void guardarDatos(){
    try {
        String apellido = txtApellidos.getText().trim();
        String nombre = txtNombre.getText().trim();
        String promedioStr = txtPromedio.getText().trim();
        if (apellido.isEmpty() || nombre.isEmpty() || promedioStr.isEmpty()) {
```

```
                JOptionPane.showMessageDialog(null, "Completa todos los campos antes de
guardar");
                return;
            }
            double promedio = Double.parseDouble(promedioStr);
            String asignatura = cboAsignatura.getSelectedItem().toString();
            String estado = cboEstado.getSelectedItem().toString();
            String SQL = "INSERT INTO estudiante (estu_apellido, estu_nombre,
estu_asignatura, estu_promedio, estu_estado) VALUES (?, ?, ?, ?, ?)";
            PreparedStatement pst=(PreparedStatement) con.prepareStatement(SQL);
            pst.setString(1, txtNombre.getText());
            pst.setString(2, txtApellidos.getText());
            int seleccion=cboAsignatura.getSelectedIndex();
            pst.setString(3,cboAsignatura.getItemAt(seleccion));
            pst.setString(4,txtPromedio.getText());
            int seleccion2=cboEstado.getSelectedIndex();
            pst.setString(5,cboEstado.getItemAt(seleccion2));
            pst.execute();
             JOptionPane.showMessageDialog(null, "Registro exitoso");
            limpiarDatos();
            // Mostramos nuevamente los datos actualizados en la tabla
            mostrarDatos();
        } catch (NumberFormatException e) {
```

```
                JOptionPane.showMessageDialog(null, "El valor del promedio debe ser numérico");
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "Error al guardar el registro: " +
e.getMessage());
            }
        }
        rivate void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
                actualizarDatos();
            }

            public void actualizarDatos(){
                try {
                int filaSeleccionada = tablaEstudiante.getSelectedRow();
                if (filaSeleccionada == -1) {
                    JOptionPane.showMessageDialog(null, "Selecciona un registro para actualizar");
                    return;
                }
                String estu_codigo = (String) tablaEstudiante.getValueAt(filaSeleccionada, 0);
                String apellido = txtApellidos.getText().trim();
                String nombre = txtNombre.getText().trim();
                String promedioStr = txtPromedio.getText().trim();
                if (apellido.isEmpty() || nombre.isEmpty() || promedioStr.isEmpty()) {
```

```
                JOptionPane.showMessageDialog(null, "Completa todos los campos antes de
actualizar");
                return;
            }
            double promedio = Double.parseDouble(promedioStr);
            String asignatura = cboAsignatura.getSelectedItem().toString();
            String estado = cboEstado.getSelectedItem().toString();
            String SQL = "UPDATE estudiante SET estu_apellido=?, estu_nombre=?,
estu_asignatura=?, estu_promedio=?, estu_estado=? WHERE estu_codigo=?";

            String dao=(String)tablaEstudiante.getValueAt(filaSeleccionada,0);
            PreparedStatement pst=(PreparedStatement) con.prepareStatement(SQL);
            pst.setString(1, apellido);
            pst.setString(2, nombre);
            pst.setString(3, asignatura);
            pst.setDouble(4, promedio);
            pst.setString(5, estado);
            pst.setString(6, estu_codigo);
            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Se ha actualizado correctamente el
registro");
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El valor del promedio debe ser numérico");
```

```
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "Error en la actualización: " +
e.getMessage());
            }
          }
        private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
            eliminarRegistro();
            }

            public void eliminarRegistro(){
                int filaSeleccionada=tablaEstudiante.getSelectedRow();
                try {
                    String estu_codigo = (String) tablaEstudiante.getValueAt(filaSeleccionada, 0);
                    String SQL="DELETE FROM estudiante where
estu_codigo="+tablaEstudiante.getValueAt(filaSeleccionada, 0);
                    Statement st=(Statement) con.createStatement();
                    int n=st.executeUpdate(SQL);
                    if(n>=0){
                        JOptionPane.showMessageDialog(null, "Registro Eliminado correctamente");
                    }
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null,"Error en eliminar
registro"+e.getMessage());
```

```
            }
        }
    public void mostrarDatos(){
        //utilizamos un titulo para llamar a nuestro arreglo en una tabla
        String
titulos[]={"Codigo","Apellido","Nombre","Asignatura","Promedio","Estado"};
        //colocamos la dimencion del arreglo que es [6];
        String registro[]=new String[6];
        DefaultTableModel modelo=new DefaultTableModel(null,titulos);
        String SQL="SELECT * FROM `estudiante`";
        try{
            Statement st=(Statement) con.createStatement();
            ResultSet rs=st.executeQuery(SQL);
            while(rs.next()){
                //llamamos a nuestros atributos que tenemos en nuestra base de datos de
localhost
                registro[0]=rs.getString("estu_codigo");
                registro[1]=rs.getString("estu_nombre");
                registro[2]=rs.getString("estu_apellido");
                registro[3]=rs.getString("estu_asignatura");
                registro[4]=rs.getString("estu_promedio");
                registro[5]=rs.getString("estu_estado");
                modelo.addRow(registro);
```

```
            }
            //llamamos a nuestra tabla que seba a registar todo los ingresos
            tablaEstudiante.setModel(modelo);
        }catch( Exception e){
            //si la informacion es incorrecta saldra que hay un error de los datos
            JOptionPane.showMessageDialog(null,"error mostrar datos"+e.getMessage());
        }
    }
```

3.6.4. Código For_profesor

```
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;
import controller.Conexion;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;



public class Form_profe extends javax.swing.JInternalFrame {
    Conexion cc=new Conexion();
    Connection con=cc.conexion();

    public Form_profe() {
```

```
            initComponents();
            mostrarDatos();
        }
        public void mostrarDatos(){

            String
titulos[]={"Codigo","Nombre","Apellido","Celular","mail","Titulo","Estado"};
            String registro[]=new String[7];
            DefaultTableModel modelo=new DefaultTableModel(null,titulos);
            String SQL="SELECT * FROM `profesor`";
            try{
                Statement st=(Statement) con.createStatement();
                ResultSet rs=st.executeQuery(SQL);
                while(rs.next()){
                    registro[0]=rs.getString("pro_codigo");
                    registro[1]=rs.getString("pro_nombre");
                    registro[2]=rs.getString("pro_apellido");
                    registro[3]=rs.getString("pro_celular");
                    registro[4]=rs.getString("pro_mail");
                    registro[5]=rs.getString("pro_titulo");
                    registro[6]=rs.getString("pro_estado");
                    modelo.addRow(registro);
                }
```

```
            tablaProfesor.setModel(modelo);
        }catch( Exception e){
            JOptionPane.showMessageDialog(null,"error mostrar datos"+e.getMessage());
        }
    }
```

3.6.5. Código For_horario

```
private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
    actualizasDatos();
    }
public void actualizasDatos(){
  try {
        int filaSeleccionada = tablaProfesor.getSelectedRow();
        if (filaSeleccionada == -1) {
            JOptionPane.showMessageDialog(null, "Selecciona un registro para actualizar");
            return;
        }
        String pro_codigo = (String) tablaProfesor.getValueAt(filaSeleccionada, 0);
        String nombre = txtNombres.getText().trim();
        String apellido = txtApellidos.getText().trim();
        String celular = txtCelular.getText().trim();
        String mail = txtmail.getText().trim();
        if (nombre.isEmpty() || apellido.isEmpty() || celular.isEmpty() || mail.isEmpty()) {
```

```
                JOptionPane.showMessageDialog(null, "Completa todos los campos antes de
actualizar");
                return;
            }
            String titulo = bnoTitulo.getSelectedItem().toString();
            String estado = bnoEstados.getSelectedItem().toString();
            String SQL = "UPDATE profesor SET pro_nombre=?, pro_apellido=?,
pro_celular=?, pro_mail=?, pro_titulo=?, pro_estado=? WHERE pro_codigo=?";
            String dao=(String)tablaProfesor.getValueAt(filaSeleccionada,0);
            PreparedStatement pst=(PreparedStatement) con.prepareStatement(SQL);
            pst.setString(1, nombre);
            pst.setString(2, apellido);
            pst.setString(3, celular);
            pst.setString(4, mail);
            pst.setString(5, titulo);
            pst.setString(6, estado);
            pst.setString(7, pro_codigo);
            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Se ha actualizado correctamente el
registro");
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El valor del promedio debe ser numérico");
        } catch (Exception e) {
```

```
            JOptionPane.showMessageDialog(null, "Error en la actualización: " +
e.getMessage());
        }
      }
    private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
            limpiarDatos();
        }
    public void limpiarDatos(){
        txtNombres.setText("");
        txtApellidos.setText("");
        txtCelular.setText("");
        txtmail.setText("");
        bnoTitulo.setSelectedItem(null);
        bnoEstados.setSelectedItem(null);

    }
    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
       eliminarDatos();
       }
    public void eliminarDatos(){
        int filaSeleccionada=tablaProfesor.getSelectedRow();
        try {
            String pro_codigo = (String) tablaProfesor.getValueAt(filaSeleccionada, 0);
```

```
                String SQL="DELETE FROM profesor where
pro_codigo="+tablaProfesor.getValueAt(filaSeleccionada, 0);
                Statement st=(Statement) con.createStatement();
                int n=st.executeUpdate(SQL);
                if(n>=0){
                   JOptionPane.showMessageDialog(null, "Registro Eliminado correctamente");
                }
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null,"Error en eliminar
registro"+e.getMessage());
            }
        }

        private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
            guardarDatos();
        }

      public void guardarDatos(){
        try {
            String nombre = txtNombres.getText().trim();
            String apellido = txtApellidos.getText().trim();
            String celular = txtCelular.getText().trim();
            String mail = txtmail.getText().trim();
```

```
            if (nombre.isEmpty() || apellido.isEmpty() || celular.isEmpty() || mail.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Completa todos los campos antes de
actualizar");
                return;
            }

            String titulo = bnoTitulo.getSelectedItem().toString();
            String estado = bnoEstados.getSelectedItem().toString();
            String SQL = "INSERT INTO profesor (pro_nombre, pro_apellido, pro_celular,
pro_mail, pro_titulo, pro_estado) VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement pst=(PreparedStatement) con.prepareStatement(SQL);
            pst.setString(1, txtNombres.getText());
            pst.setString(2, txtApellidos.getText());
            pst.setString(3, txtCelular.getText());
            pst.setString(4,txtmail.getText());
            int seleccion1=bnoTitulo.getSelectedIndex();
            pst.setString(5,bnoTitulo.getItemAt(seleccion1));
            int seleccion2=bnoEstados.getSelectedIndex();
            pst.setString(6,bnoEstados.getItemAt(seleccion2));
             //aqui ejecutamos la consulta
            pst.execute();
             JOptionPane.showMessageDialog(null, "Registro exitoso");
            // Limpiamos los campos después de guardar
```

```
            limpiarDatos();
            // Mostramos nuevamente los datos actualizados en la tabla
            mostrarDatos();
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El valor del promedio debe ser numérico");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error al guardar el registro: " +
e.getMessage());
        }
    }
```

3.7. Diagramas relacionales

3.7.1. Diagrama de clase

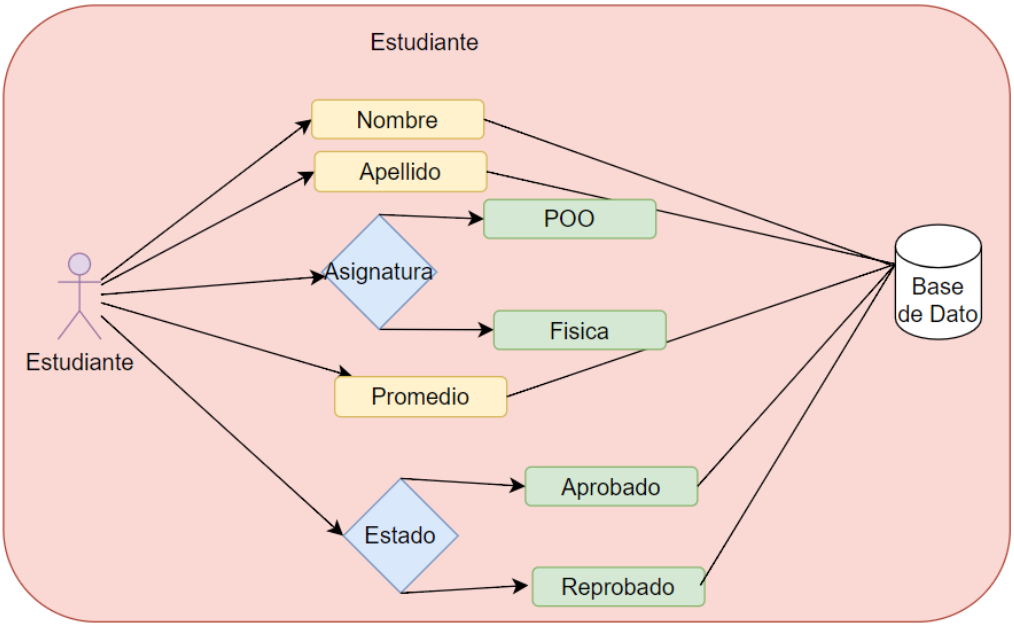
Representa las clases del sistema, sus atributos y métodos, y las relaciones entre ellas, como asociaciones, herencia y dependencias.

En este programa nos ayuda mucho para realizar interfaces que ayuden al registro de usuarios por medio de arreglos y herencia donde vamos almacenar nuestra registro por medio de una base de datos.

3.7.2. Diagrama de uso

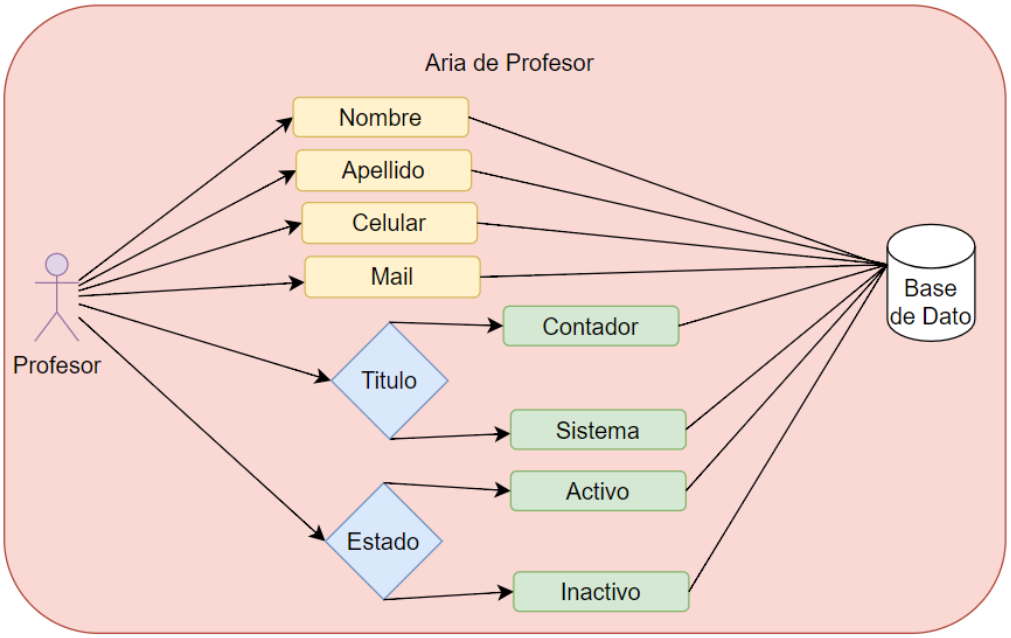
Ilustra las interacciones entre el sistema y los actores externos, describiendo los diferentes casos de uso y cómo interactúan con el sistema. Muestra la interacción entre objetos a lo largo del tiempo, enfocándose en el flujo de mensajes entre ellos.

Imagen 32. Diagrama de uso Estidiante por Kelvin Quezada



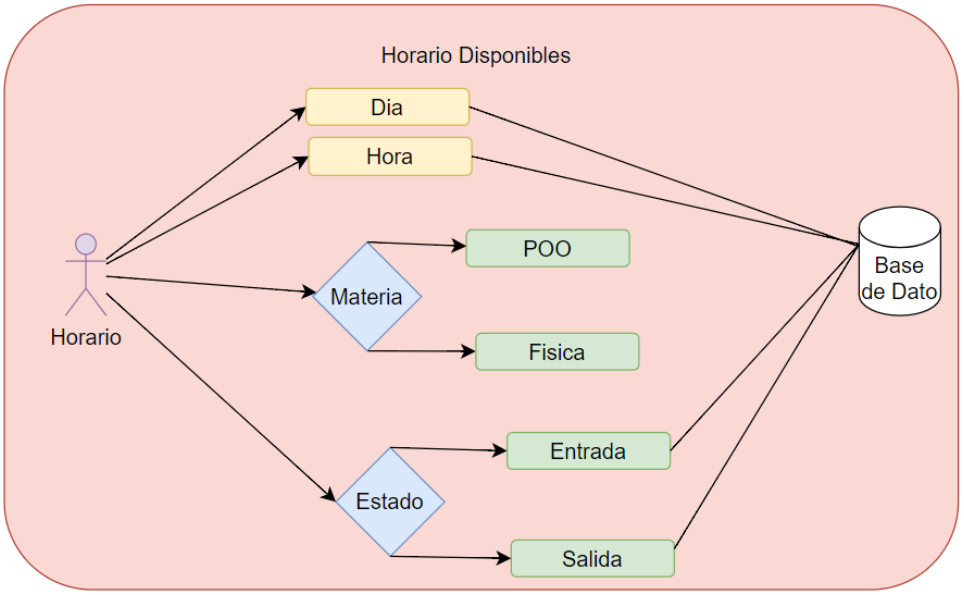
En el registro de estudiante nos dirá que ingrese los datos personales del usuario y elija la asignatura y también promedio y esa información seba almacenar en una base de datos localhost.

Imagen 33. Diagrama de uso Profesor por Kelvin Quezada



En el registro de profesores nos dirá que ingrese los datos personales del usuario y elija el título que se graduó y también elegir el estado del docente y esa información seba almacenar en una base de datos localhost.

Imagen 34. Diagrama de uso Horario por Kelvin Quezada

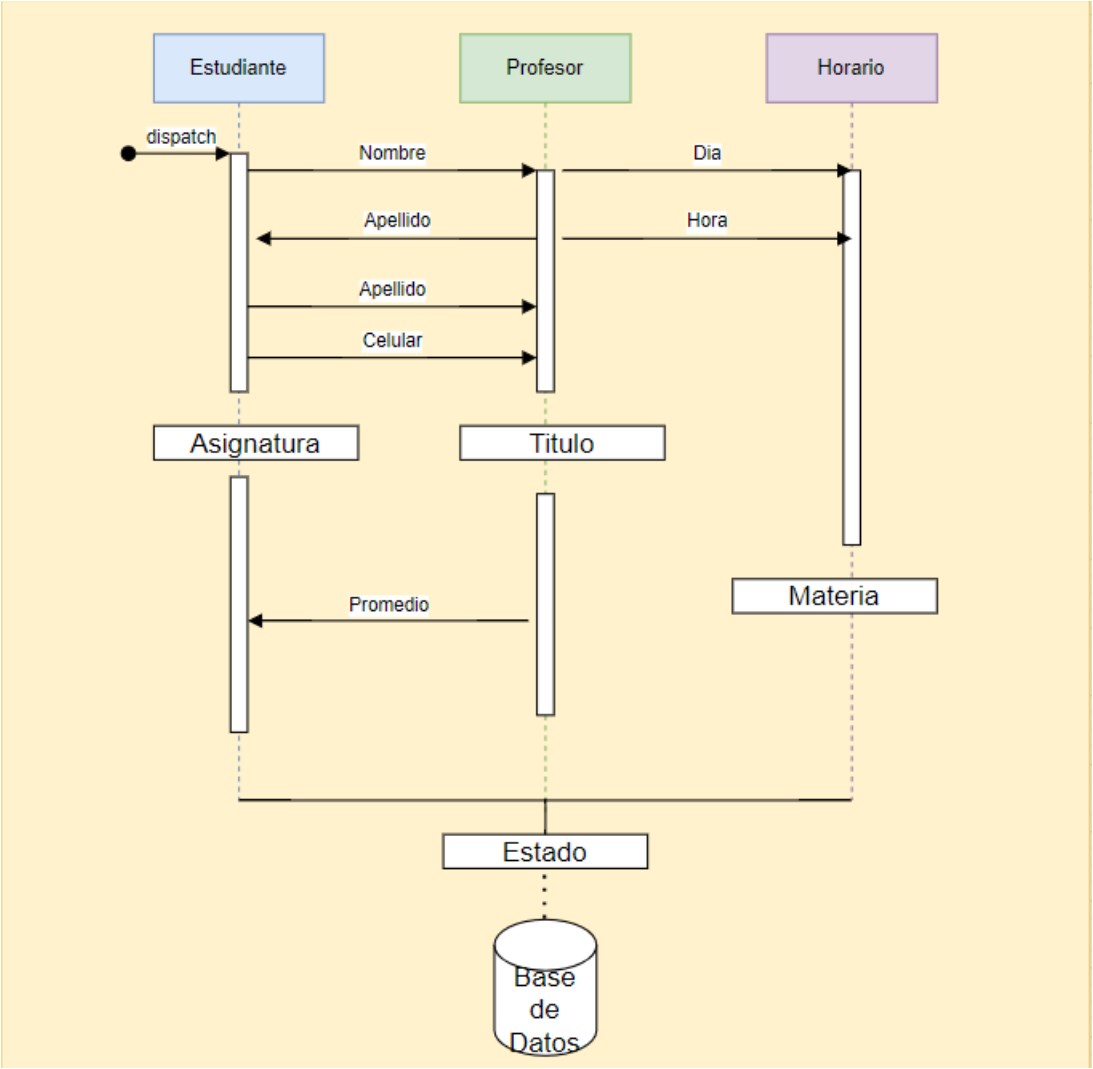


En el registro de Horarios nos dirá que ingrese los datos personales del usuario y elija la Materia y también elegir el Estado y esa información seba almacenar en una base de datos localhost.

3.7.3. Diagrama de Secuencia

Un diagrama de secuencia es mostrar la secuencia de interacciones entre los objetos en un escenario específico, capturando el flujo de mensajes y las llamadas entre ellos en un orden cronológico.

Imagen 35. Diagrama de secuencia por Kelvin Quezada



4. Conclusiones

- Las interfaces permiten definir un conjunto de métodos que ayudan a la creación de conexiones y facilite la reutilización de código por medio de las interfaces nos ayuda mucho ya que mediante las interfaces podemos crear diferentes tipos de proyectos que ayuden a la sociedad.
- Pude aprender que por medio de interfaces podemos subir nuestro registro que ingresamos en el sistema y se almacene en la base de datos.

5. Recomendaciones

- Es fundamental diseñar interfaces con un propósito claro y enfocadas en definir comportamientos relacionados.
- Desarrollar habilidades que puedes generar mediante interfaces puedes crear juegos y realizar un registro con base de datos que pueden ayudar a las empresas.

6. Bibliografía

Canelo, M. M. (02 de Noviembre de 2020). *profile*. Obtenido de profile: https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

Dataprix. (29 de Septiembre de 2009). *Generalizacion/especializacion*. Obtenido de Generalizacion/especializacion: https://www.dataprix.com/es/bases-datos-master-software-libre-uoc/221-generalizacionespecializacion#:~:text=La%20generalizaci%C3%B3n%2Fespecializaci%C3%B3n%20permite%20reflejar,vista%20de%20una%20forma%20gen%C3%A9rica.

Jesús. (25 de Abril de 2022). *DONGEE*. Obtenido de DONGEE: https://www.dongee.com/tutoriales/que-es-xampp/

Mancuzo, G. (24 de Junio de 2021). *comparasoftware*. Obtenido de comparasoftware: https://blog.comparasoftware.com/diagramas-de-uml-que-significa-esta-metodologia/

Cabrera, I. (2022, January 12). *Todo lo que necesitas saber sobre el diagrama de caso de uso*. Venngage. Retrieved May 30, 2023, from https://es.venngage.com/blog/diagrama-de-caso-de-uso/

Lara, D. (2015, July 7). *Encapsulamiento en la programación orientada a objetos*. Styde.net. Retrieved May 30, 2023, from https://styde.net/encapsulamiento-en-la-programacion-orientada-a-objetos/

Mancuzo, G. (2021, June 24). *Qué son los Diagramas de UML? + Tipos + Importancia*. BlogComparaSoftware. Retrieved May 30, 2023, from https://blog.comparasoftware.com/diagramas-de-uml-que-significa-esta-metodologia/

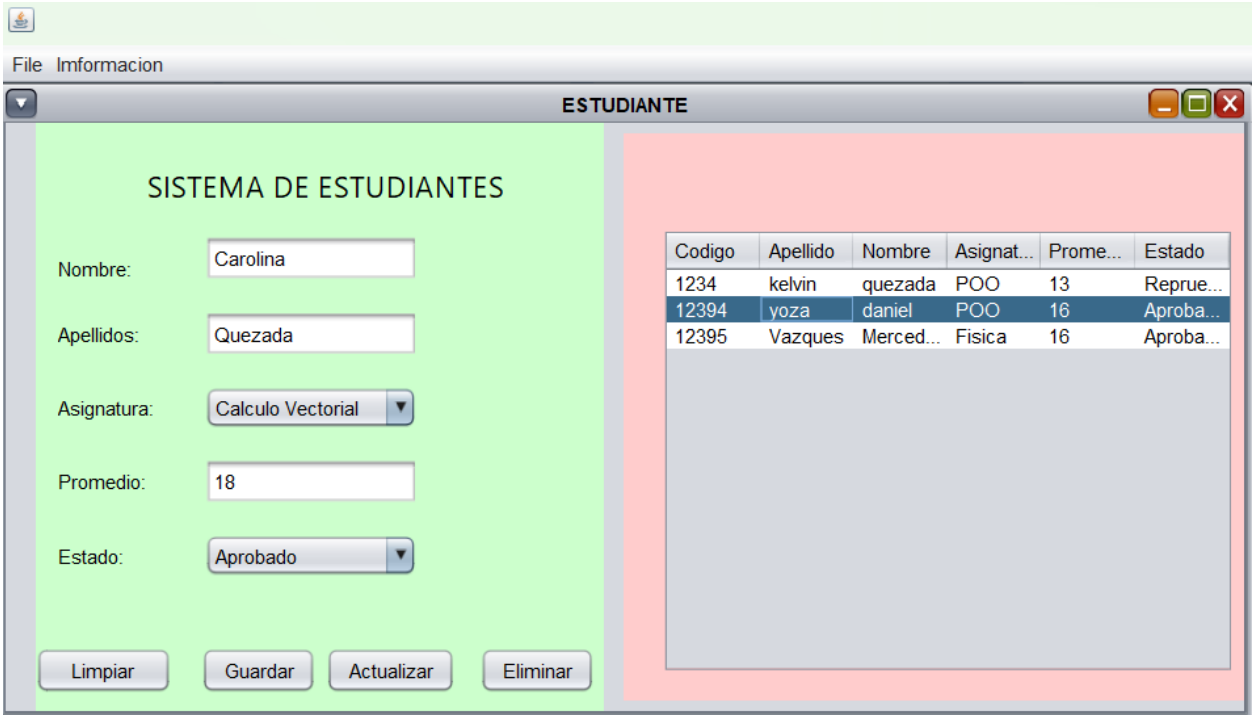
Martínez, M. (2020, November 2). *¿Qué es la Programación Orientada a Objetos?* Profile. Retrieved May 30, 2023, from https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

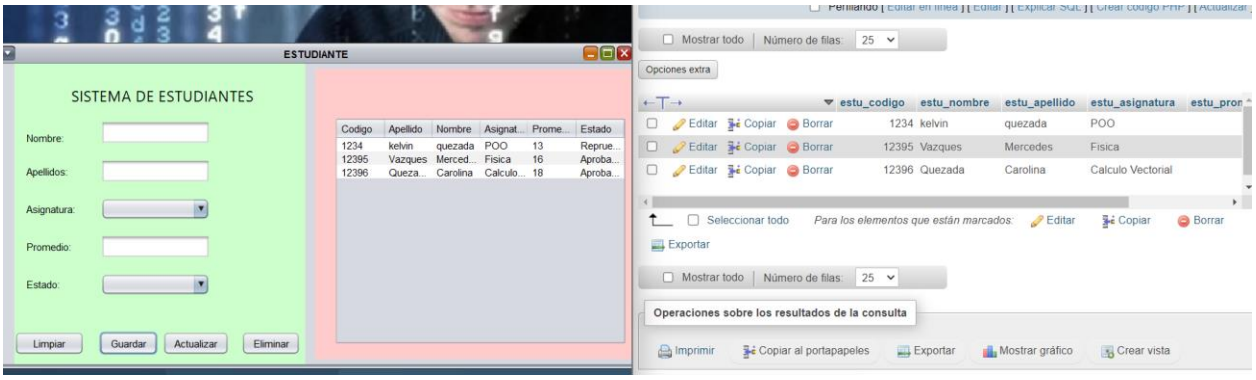
Pantoja, B. (2004). *El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing*. SciELO Bolivia. Retrieved June 24, 2023, from http://www.scielo.org.bo/scielo.php?pid=S1683-07892004000100005&script=sci_arttext

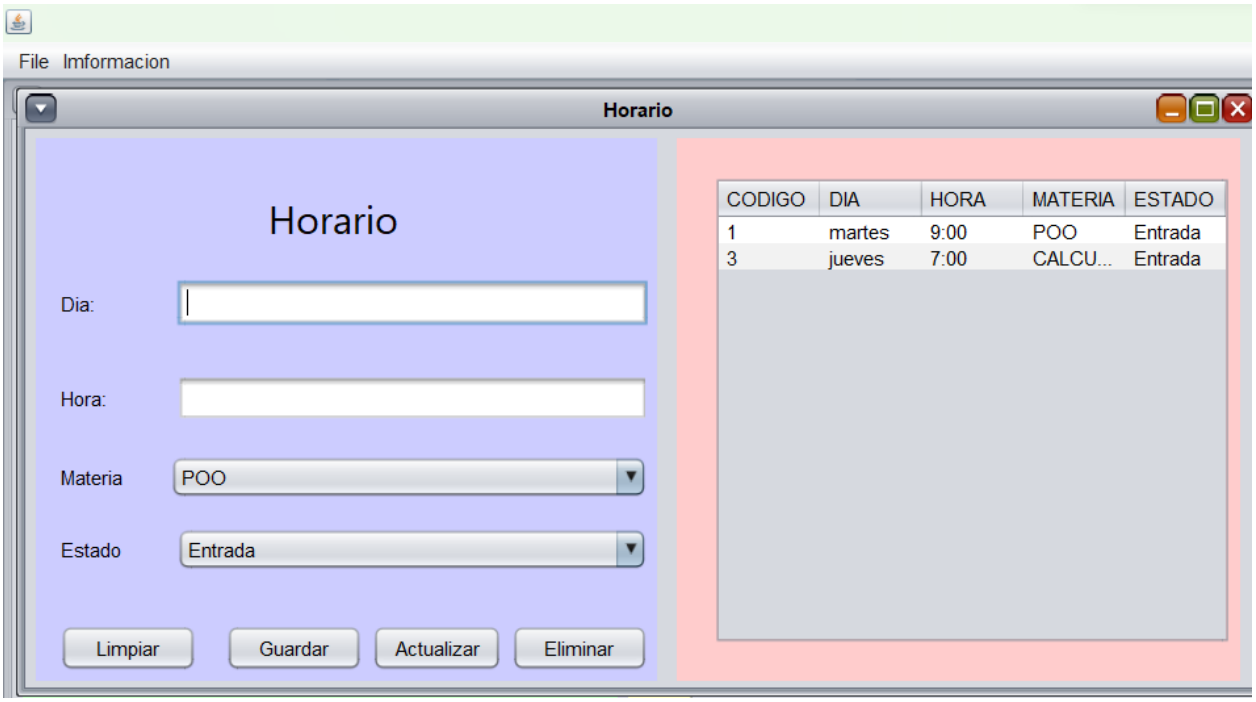
7. Anexos

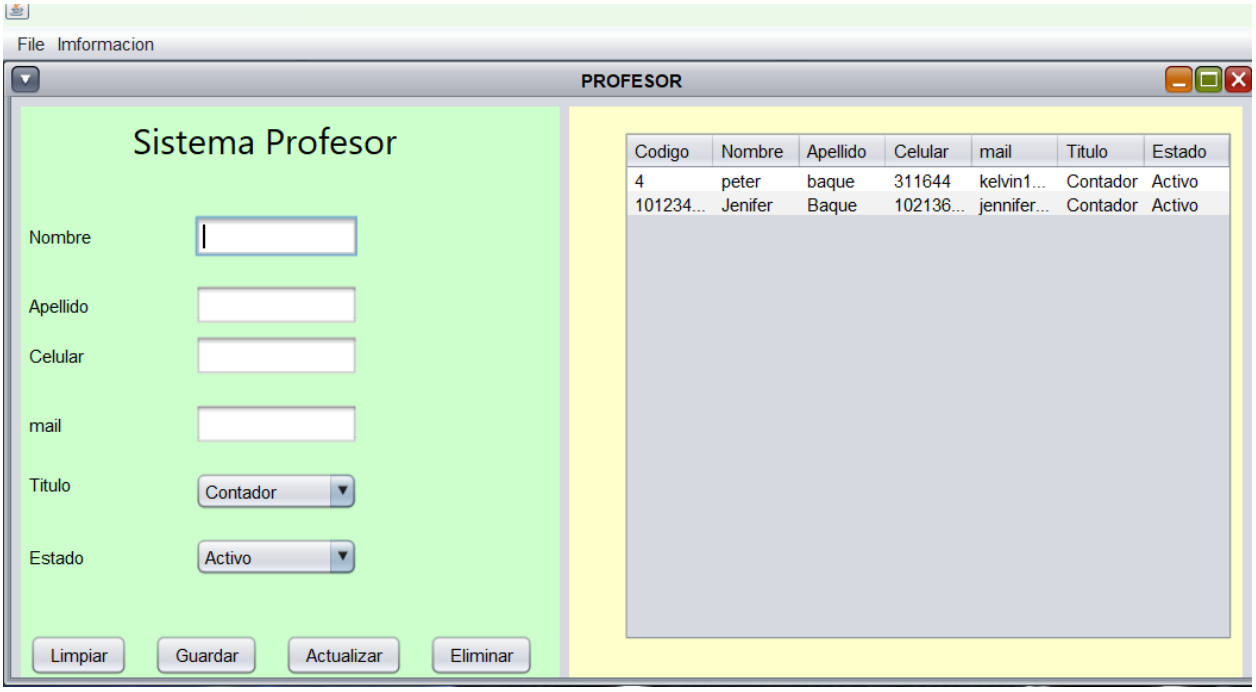

File Imformacion
ESTUDIANTE
SISTEMA DE ESTUDIANTES
Nombre:
Mercedes
Apellidos:
Vazques
Asignatura:
Fisica
Promedio:
16
Estado:
Aprobado
Limpiar
Guardar
Actualizar
Eliminar
Codigo Apellido Nombre Asignat... Prome... Estado
1234 kelvin quezada POO 13 Reprue...
12394 yoza daniel POO 16 Aproba...


ESTUDIANTE
SISTEMA DE ESTUDIANTES
Nombre:
Apellidos:
Asignatura:
Promedio:
Estado:
Limpiar
Guardar
Actualizar
Eliminar
Codigo Apellido Nombre Asignat... Prome... Estado
1234 kelvin quezada POO 13 Reprue...
12394 yoza daniel POO 16 Aproba...
12395 Vazques Merced... Fisica 16 Aproba...
Message
Registro Eliminado correctamente
OK


File Imformacion
ESTUDIANTE
SISTEMA DE ESTUDIANTES
Nombre:
Carolina
Apellidos:
Quezada
Asignatura:
Calculo Vectorial
Promedio:
18
Estado:
Aprobado
Limpiar
Guardar
Actualizar
Eliminar
Codigo Apellido Nombre Asignat... Prome... Estado
1234 kelvin quezada POO 13 Reprue...
12394 yoza daniel POO 16 Aproba...
12395 Vazques Merced... Fisica 16 Aproba...


ESTUDIANTE
SISTEMA DE ESTUDIANTES
Nombre:
Apellidos:
Asignatura:
Promedio:
Estado:
Limpiar
Guardar
Actualizar
Eliminar
Mostrar todo
Número de filas:
Opciones extra
estu_codigo estu_nombre estu_apellido estu_asignatura
Editar Copiar Borrar 1234 kelvin quezada POO
Editar Copiar Borrar 12395 Vazques Mercedes Fisica
Editar Copiar Borrar 12396 Quezada Carolina Calculo Vectorial
Seleccionar todo
Para los elementos que están marcados:
Exportar
Operaciones sobre los resultados de la consulta
Imprimir
Copiar al portapapeles
Exportar
Mostrar gráfico
Crear vista


File Imformacion
Horario
Horario
Dia:
Hora:
Materia
POO
Estado
Entrada
Limpiar
Guardar
Actualizar
Eliminar
CODIGO DIA HORA MATERIA ESTADO
1 martes 9:00 POO Entrada
3 jueves 7:00 CALCU... Entrada


File Imformacion
PROFESOR
Sistema Profesor
Nombre
Apellido
Celular
mail
Titulo
Contador
Estado
Activo
Limpiar
Guardar
Actualizar
Eliminar
Codigo Nombre Apellido Celular mail Titulo Estado
4 peter baque 311644 kelvin1... Contador Activo
101234... Jenifer Baque 102136... jennifer... Contador Activo