



TECNOLÓGICO NACIONAL DE MÉXICO  
INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA

MATERIA:

ADMINISTRACIÓN Y ORGANIZACIÓN DE DATOS

IFF-1003

DOCENTE:

RAMIREZ SANTIAGO BENEDICTO

ESTUDIANTE:

RUIZ LÓPEZ KAREN MAGALI

ACTIVIDAD:

A2.3 PROYECTO EN JAVA DE ARCHIVOS CON ORGANIZACIÓN DIRECTA

UNIDAD 3

ORGANIZACIONES BÁSICAS

CARRERA:

INGENIERÍA INFORMÁTICA

4 SEMESTRE

GRUPO: 4AI

DEPARTAMENTO DE CIENCIAS ECONÓMICO-ADMINISTRATIVO

FECHA DE ENTREGA: 13/05/2025

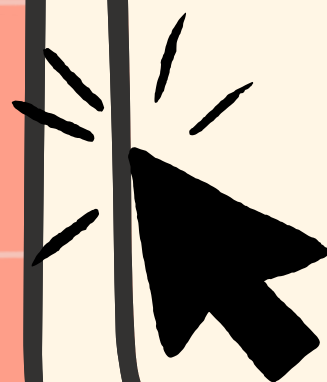
ENERO - JUNIO 2025

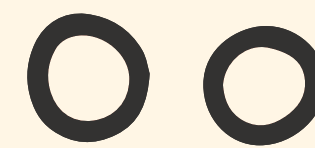
NAZARENO, SANTA CRUZ XOXOCOTLÁN, OAXACA



# ÍNDICE

INTRODUCCIÓN.....	3
ControladorCRUDEstudiantes.....	4
ArchivoOrgnDir.....	10
• VistaCRUDEstudiantes.....	23
• FrmAgregarEstudiante.....	29
• FrmEliminarEstudiante.....	35
• FrmModificarEstudiante.....	38
• Ejecución de proyecto.....	47
CONCLUSIÓN.....	52



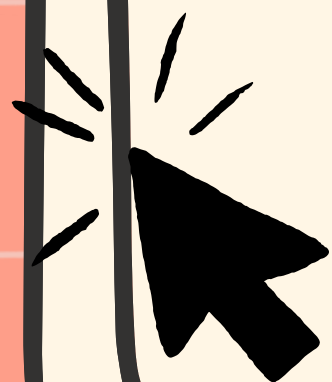


# INTRODUCCIÓN

El presente sistema ha sido desarrollado con el objetivo de gestionar de manera eficiente el registro, búsqueda, modificación y eliminación de estudiantes. Para ello, se ha implementado un CRUD (Crear, Leer, Actualizar y Eliminar) utilizando el lenguaje de programación Java y el paradigma de programación orientado a objetos.

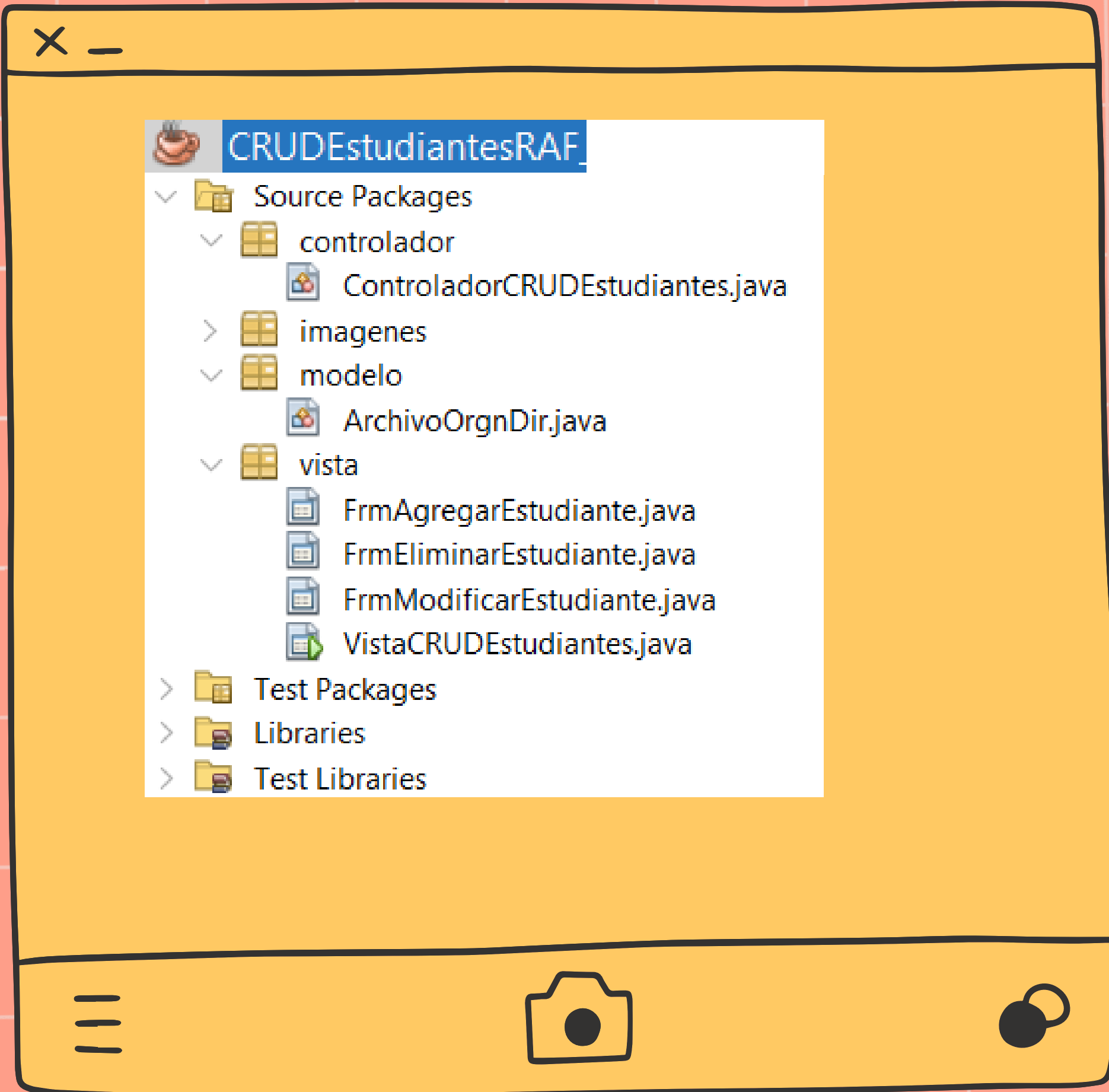
El sistema se estructura en tres capas fundamentales:

1. Vista: Encargada de la interfaz gráfica del usuario (GUI), permitiendo la interacción con el sistema mediante formularios amigables e intuitivos. Se utilizaron componentes de Swing como JFrame, JTextField, JComboBox y JRadioButton.
2. Controlador: Actúa como puente entre la vista y el modelo, gestionando la lógica de negocio. Se encarga de recibir los datos ingresados por el usuario y coordinar su procesamiento y almacenamiento.
3. Modelo: Responsable del manejo de los datos, en este caso a través de archivos binarios con organización por dirección utilizando RandomAccessFile. Incluye clases como ArchivoOrgnDir, que permiten acceder, modificar o eliminar registros sin cargar todo el archivo en memoria.



# CONTROLADOR CRUD ESTUDIANTES

En esta parte del proyecto desarrollé el controlador `ControladorCRUDEstudiantes`, que se conecta con la interfaz gráfica y la clase de manejo de archivos para implementar un CRUD completo de estudiantes usando archivos binarios con acceso directo.



```
//Controlador
package controlador;
import javax.swing.table.DefaultTableModel;
import modelo.ArchivoOrgnDir;
import vista.VistaCRUDEstudiantes;
public class ControladorCRUDEstudiantes {
    public VistaCRUDEstudiantes objVistaCRUDEst; //se usa para acceder a la interfaz gráfica.
    private ArchivoOrgnDir objArchivo; //maneja las operaciones con archivos (lectura, escritura, e
    private DefaultTableModel modelo; //estructura que representa los datos para una tabla (JTable)
    public ControladorCRUDEstudiantes(VistaCRUDEstudiantes vista, ArchivoOrgnDir archivo) {
        this.objVistaCRUDEst = vista;
        this.objArchivo = archivo;
        this.modelo = null;
        /*Asocia el controlador con una vista y un archivo.
        Inicializa el modelo de tabla en null.*/
    }
}
```

Este controlador es la parte lógica del sistema CRUD de estudiantes. Se encarga de:

- Conectarse con la interfaz gráfica (VistaCRUDEstudiantes) para mostrar los datos.
- Manipular el archivo binario estudiantes.dat con ayuda de la clase ArchivoOrgnDir, que usa RandomAccessFile.
- Controlar la lectura, escritura, búsqueda, eliminación y edición de registros.
- Actualizar automáticamente la tabla visual cada vez que se realiza alguna operación.



# ¿QUÉ HACE LLENAR TABLA?

```
public class ControladorCRUDEstudiantes {
    public VistaCRUDEstudiantes objVistaCRUDEst; //se usa para acceder a la interfaz gráfica.
    private ArchivoOrgnDir objArchivo; //maneja las operaciones con archivos (lectura, escritura, etc.)
    private DefaultTableModel modelo; //estructura que representa los datos para una tabla (JTable).
    public ControladorCRUDEstudiantes(VistaCRUDEstudiantes vista, ArchivoOrgnDir archivo) {
        this.objVistaCRUDEst = vista;
        this.objArchivo = archivo;
        this.modelo = null;
        /*Asocia el controlador con una vista y un archivo.
        Inicializa el modelo de tabla en null.*/
    }
    public void llenarTabla() {
        String[] columnas = {"Num.Control", "Nombre", "Apellidos", "Semestre", "Grupo", "Carrera"};
        //Define los nombres de las columnas para la tabla.
        this.objArchivo = new ArchivoOrgnDir();
        //Crea una nueva instancia del manejador de archivos.
        if (this.objArchivo.abrirArchivoRAF(nombreArchivo: "estudiantes.dat")) {
            //Abre el archivo estudiantes.dat con acceso aleatorio.
            String[][] filas = this.objArchivo.obtenerMatrizRegistros(numCol: columnas.length);
            //Obtiene una matriz con los registros del archivo (tantas columnas como las definidas).
            this.objArchivo.cerrarArchivo();
            //Cierra el archivo.
            if (filas != null) {
                modelo = new DefaultTableModel(data: filas, columnNames: columnas);
                this.objVistaCRUDEst.jtblEstudiantes.setModel(dataModel: modelo);
            } else {
                System.out.println(x: "No se pudieron leer los registros del archivo.");
            } //Mensajes de error si no se pudo leer o abrir el archivo.
        } else {
            System.out.println(x: "Error al abrir el archivo para llenar la tabla.");
        } //Si se leyeron registros, se crea un modelo de tabla y se muestra en la vista.
    }
}
```

- Constructor: Asocia el controlador con la vista (VistaCRUDEstudiantes) y el archivo (ArchivoOrgnDir). Inicializa el modelo de la tabla.
- llenarTabla(): Abre el archivo estudiantes.dat, lee todos los registros válidos, y los carga en la tabla de la vista. Si no hay datos o falla el acceso, muestra un error en consola.

# ¿QUÉ HACEN ESTOS MÉTODOS?

```
public void guardarRegistro(String nc, String nom, String ape, int sem, char gpo, String carrera) {
    this.objArchivo = new ArchivoOrgnDir();
    //Se crea una nueva instancia para escribir datos.
    if (this.objArchivo.abrirArchivoRAF(nombreArchivo: "estudiantes.dat")) {
        //Abre el archivo de estudiantes.
        this.objArchivo.escribirRegistro(numControl: nc, nombre: nom, apellidos: ape, semestre: sem, grupo: gpo, carrera);
        //Escribe un nuevo registro con los datos recibidos.
        this.objArchivo.cerrarArchivo();
        this.llenarTabla();
        //Cierra el archivo y actualiza la tabla de la vista con los nuevos datos.
    } else {
        System.out.println(x: "Error al abrir el archivo para guardar.");
    }
}

public String[] buscarRegistro(String nc) {
    /*Se inicializa una variable para guardar los datos encontrados.
    crea una nueva instancia del archivo.*/

    String[] datos = null;
    this.objArchivo = new ArchivoOrgnDir();

    if (this.objArchivo.abrirArchivoRAF(nombreArchivo: "estudiantes.dat")) {
        datos = this.objArchivo.buscarDato(nc);
        //Abre el archivo y busca el dato con el número de control (nc).
        this.objArchivo.cerrarArchivo();
    } else {
        System.out.println(x: "Error al abrir el archivo para buscar.");
    }

    return datos; //Cierra el archivo y devuelve el arreglo de datos del estudiante encontrado (o null si no se encuentra)
}
```

- **guardarRegistro(...):** Recibe los datos de un nuevo estudiante, abre el archivo, escribe el registro en una posición libre (o nueva), lo guarda y actualiza la tabla.
- **buscarRegistro(nc):** Abre el archivo y busca un estudiante con el número de control dado. Devuelve los datos del estudiante o null si no lo encuentra.



# ¿QUÉ HACEN ESTOS MÉTODOS?

```
public boolean validaNumControl(String numControl) {
    //Verifica si el número de control cumple el formato correcto usando una expresión regular (8 dígitos).
    this.objArchivo = new ArchivoOrgnDir();
    return this.objArchivo.validaControl(numControl);
}

public void eliminarRegistro(String nc) {
    this.objArchivo = new ArchivoOrgnDir();
    //Abre el archivo y "borra" el registro con ese número de control (probablemente sobrescribiendo con "00000000").
    if (this.objArchivo.abrirArchivoRAF(nombreArchivo: "estudiantes.dat")) {
        this.objArchivo.eliminarLinea(numControl: nc);
        this.objArchivo.cerrarArchivo();
        this.llenarTabla();
    } else {
        System.out.println(x: "Error al abrir el archivo para eliminar.");
    } //Actualiza la tabla para reflejar los cambios.
}

public void editarRegistro(String nc, String nuevoNom, String nuevoApe, int nuevoSem, char nuevoGpo, String nuevaCarrera) {
    this.objArchivo = new ArchivoOrgnDir();
    //Prepara los nuevos datos para ser escritos.
    if (this.objArchivo.abrirArchivoRAF(nombreArchivo: "estudiantes.dat")) {
        this.objArchivo.escribirRegistro(numControl: nc, nombre: nuevoNom, apellidos: nuevoApe, semestre: nuevoSem, grupo: nuevoGpo,
        //Escribe los nuevos datos en la posición del mismo número de control (sobrescribe).
        this.objArchivo.cerrarArchivo();
        //Cierra el archivo y actualiza la tabla para reflejar los cambios.
        this.llenarTabla();
    } else {
        System.out.println(x: "Error al abrir el archivo para editar.");
    }
}
```

- **validaNumControl(nc):**  
Verifica que el número de control tenga exactamente 8 dígitos. Esto es útil para validar la entrada del usuario.
- **eliminarRegistro(nc):** Marca como eliminado el registro que coincide con ese número de control. Generalmente se hace escribiendo ceros en esa posición. Luego actualiza la tabla.



# RESUMEN DE LO QUE HACE

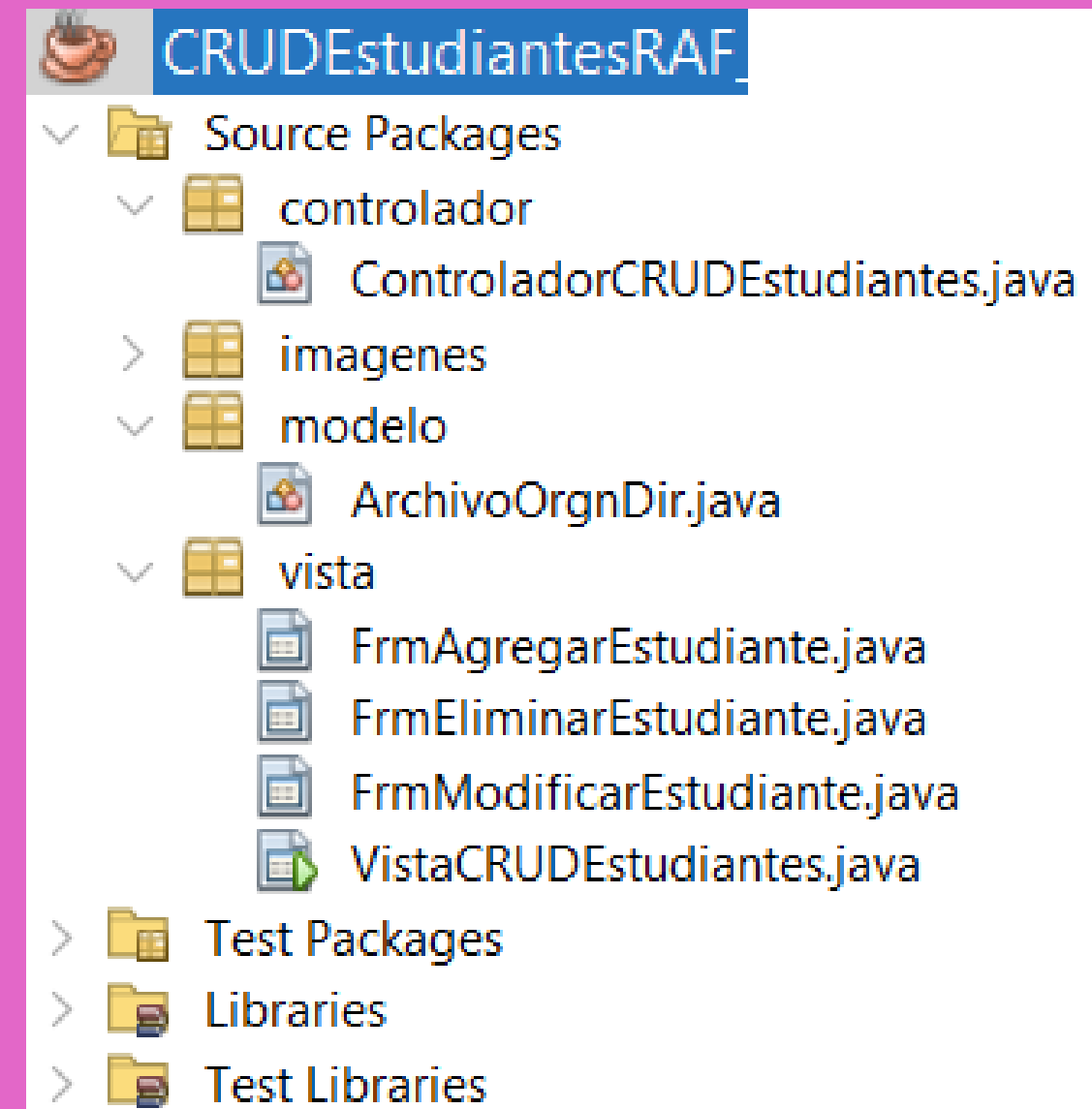
- `llenarTabla()` Lee todos los registros y los muestra en la tabla de la vista.
  - `guardarRegistro(...)` Guarda un nuevo estudiante en el archivo y actualiza la tabla.
  - `buscarRegistro(nc)` Devuelve los datos de un estudiante según su número de control.
  - `validaNumControl(...)` Verifica que el número de control tenga 8 dígitos.
  - `eliminarRegistro(nc)` Marca un registro como eliminado y actualiza la tabla.
- `editarRegistro(...)` Sobrescribe un registro existente con datos nuevos y actualiza la tabla.

# CLASE ARCHIVO ORGN DIR

Objetivo:

Gestionar registros de estudiantes en un archivo binario con acceso directo, utilizando el número de control como clave única.

El uso de tamaños fijos es clave para lograr acceso directo, ya que permite calcular fácilmente la posición de cualquier registro.



```

/*Organización directa: permite acceder directamente a una posición del archivo sin tener que leerlo completo.
Esto se logra calculando la posición en bytes donde debe escribirse o leerse un registro, basado en una clave
única (como el número de control).
trim: método de la clase String que elimina los espacios en blanco al inicio y al final de una cadena de texto.*/
package modelo;//La clase pertenece al paquete modelo, que normalmente agrupa la lógica de datos o negocio.
import java.io.*;// Se importa todo lo necesario para manejar archivos (como File, RandomAccessFile, IOException)
import java.text.DecimalFormat;//Se usa para formatear números, en este caso el semestre (como "01", "02"...).
import java.util.logging.Level;//Sirve para definir la severidad de los mensajes que se van a registrar en el log del sistema
import java.util.logging.Logger;//Para registrar errores cuando algo falla con el archivo.
public class ArchivoOrgnDir { //Se define la clase que manejará archivos binarios con organización directa.
    private File fichero;//Representa el archivo físico.
    private RandomAccessFile raf; /*Permite leer y escribir datos en cualquier posición del archivo
                                   (esto es lo que permite la "organización directa").*/
    //Tamaños fijos para cada campo de un registro
    private final int NUMCONTROL_LEN = 8;//
    private final int NOMBRE_LEN = 20;//
    private final int APELLIDOS_LEN = 20;//
    private final int SEMESTRE_LEN = 2;//
    private final int GRUPO_LEN = 1;
    private final int CARRERA_LEN = 30;
    //TAMREG: tamaño total en bytes de cada registro. Esto es clave para calcular en qué posición se debe leer o escribir un regi
    private final int TAMREG = NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN + SEMESTRE_LEN + GRUPO_LEN + CARRERA_LEN;
    //Se usa para saber cuántos registros hay en total en el archivo.
    private int totReg;

```

### Atributos de longitud fija:

- Cada campo (como número de control, nombre, apellidos, etc.) tiene una longitud específica en bytes. Por ejemplo, el número de control ocupa 8 bytes, el nombre 20, etc. La suma total de estos define el tamaño de un registro completo (TAMREG).



/\* Constructor

Funcion:Inicializa las variables. El archivo aún no está abierto ni creado.\*/

```
public ArchivoOrgnDir() { //Este constructor inicializa los atributos de la clase cuando se crea un objeto de tipo ArchivoOrgnDir.
```

```
    this.fichero = null; //fichero es una variable de tipo File, que representa un archivo físico en el disco.
```

/\*Inicializa el atributo fichero como null, lo que significa que aún no se ha asociado con ningún archivo real.

Es una buena práctica para indicar que no se está trabajando con ningún archivo todavía. \*/

```
    this.raf = null; //Es una variable de tipo RandomAccessFile, una clase que permite leer y escribir en cualquier parte de un archivo
```

/\*Inicializa la variable raf como null porque todavía no se ha abierto ningún archivo con acceso aleatorio.\*/

```
    this.totReg = 0; //Es un contador entero que guarda el total de registros en el archivo.
```

//Lo inicia en 0 porque aún no se han contado ni leído registros del archivo.

```
}
```

//Este método prepara el archivo con el que se quiere trabajar, y verifica si ya existe en el sistema de archivos.

```
private boolean establecerFlujo(String nombreArchivo) {
```

//Crea un objeto File que representa un archivo con el nombre o ruta que se pasa como parámetro (nombreArchivo).

```
    fichero = new File(pathname: nombreArchivo);
```

/\*Asigna a fichero una referencia a ese archivo. No lo abre ni lo crea todavía, solo establece un objeto File que apunta a ese archivo.

Para que las operaciones posteriores (abrir, leer, escribir, borrar) sepan a qué archivo se refieren.\*/

```
    return fichero.exists(); //Verifica si el archivo ya existe en el disco.
```

/\*Devuelve true si el archivo existe, o false si no. Es útil para verificar antes de intentar abrir o escribir en el archivo. \*/

```
}
```

## Cálculo de posición (establecerPosicionByte):

Dado un número de control, se calcula la posición exacta en el archivo donde debe estar el registro. Esto se hace extrayendo los últimos dígitos del número de control, convirtiéndolos a número, restando 1 (para indexar desde cero) y multiplicando por el tamaño del registro.

```

/* abrirArchivoRAF método intenta abrir el archivo representado por fichero usando la clase RandomAccessFile,
que permite leer y escribir en cualquier parte del archivo. */
public boolean abrirArchivoRAF(String nombreArchivo) {
    try { //Todas las operaciones de (entrada/salida) en Java pueden lanzar excepciones, como cuando el disco está lleno o el archivo
//Calcula en qué parte del archivo se debe escribir el registro.
        establecerFlujo(nombreArchivo);
        //Esta línea abre el archivo con el modo indicado, y se lo asigna a raf.
        this.raf = new RandomAccessFile(name: nombreArchivo, mode: "rw");

/*fichero ya debe estar apuntando a un archivo válido.
modo puede ser:
"r" solo lectura
"rw"lectura y escritura*/
        return true; // Si se pudo abrir el archivo correctamente, se retorna true.
    } catch (FileNotFoundException ex) { // Si el archivo no se puede abrir, esta parte atrapa el error.
        Logger.getLogger(ArchivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        return false; //Devuelve false si ocurrió un error (por ejemplo, si no existe el archivo o no tienes permisos).
    }
}

public void cerrarArchivo() { //Este método cierra el archivo que se estaba usando, solo si ya estaba abierto.
/*raf es un objeto de tipo RandomAccessFile que sirve para leer y escribir archivos.
raf != null verifica si el archivo está abierto. Si está cerrado, sería null, y no se hace nada.
raf.close(); cierra correctamente el archivo para liberar la memoria y evitar errores o archivos dañados.
Si ocurre un error al cerrar, lo muestra con el Logger (no detiene el programa, pero registra el error).*/
    try { //Todas las operaciones de E/S (entrada/salida) en Java pueden lanzar excepciones, como cuando el disco está lleno o el arch
//Calcula en qué parte del archivo se debe escribir el registro.
        if (raf != null) raf.close();
    } catch (IOException ex) {
        Logger.getLogger(ArchivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    } //Devuelve false si ocurrió un error (por ejemplo, si no existe el archivo o no tienes permisos).
}

```



# ¿QUÉ SE ESTA HACIENDO?

```
private static String ajustar(String s, int longitud) {  
    //Este método ajusta un texto (String) a una longitud fija, recortando o rellenando con espacios.  
    return s.length() > longitud ? s.substring(beginIndex: 0, endIndex: longitud) : String.format("%-" + longitud + "s", args: s);  
}  
/*Este método ajusta un texto (String) a una longitud fija, recortando o rellenando con espacios.  
Explicación:  
Si el texto (s) es más largo que longitud, lo recorta con substring(0, longitud).  
Si el texto es más corto, lo rellena con espacios en blanco hasta que tenga justo la longitud especificada.  
Si guardo "Jose" en un campo de 10 caracteres, esto te devuelve: "Jose "  
Si quieres guardar "GuillermoPérez" en un campo de 10 caracteres, esto te devuelve: "Guillermo"  
Esto es importante porque en archivos de longitud fija, cada campo debe ocupar el mismo espacio siempre.*/  
}  
  
private long establecerPosicionByte(String numeroControl) {  
    //Calcula en qué posición del archivo (en bytes) debe escribirse o leerse un registro específico, usando el número de control como clave.  
    return ((Long.parseLong(s: numeroControl.substring(beginIndex: 5))) - 1) * TAMREG;  
}  
/*numeroControl.substring(5) Extrae los últimos dígitos del número de control.  
Ejemplo: "23920248" se queda con "248" (desde el carácter 5 en adelante).  
Long.parseLong(...) Convierte "248" a un número: 248. (2 - 1) * 81  
-1 Se le resta 1 para que empiece desde posición 0.  
* TAMREG Se multiplica por el tamaño fijo de un registro, para saber en qué byte exacto del archivo está ese registro.*/  
}
```



# ¿Y AQUÍ?

```
//cualquier otra clase (dentro o fuera del paquete modelo) puede llamar a este método. y void que no retorna ningun valor
public void escribirRegistro(String numControl, String nombre, String apellidos, int semestre, char grupo, String carrera) {
//Este método escribe un registro completo en el archivo, en una posición específica según el número de control del estudiante.
    try { //Todas las operaciones de (entrada/salida) en Java pueden lanzar excepciones, como cuando el disco está lleno o el archivo está
//Calcula en qué parte del archivo se debe escribir el registro.
        long pos = establecerPosicionByte(numeroControl: numControl); //devuelve un índice en bytes según la fórmula de organización directa.
        raf.seek(pos); //Mueve el puntero del archivo a esa posición.
/*Luego escribe cada campo ajustado a una longitud fija (rellenado o recortado), usando codificación "ISO-8859-1"
para que los caracteres ocupen solo 1 byte (ASCII extendido).
    Los campos que se escriben:
    numControl: con longitud NUMCONTROL_LEN=8
raf.write(ajustar(...).getBytes("ISO-8859-1")): Recorta o rellena el String para que siempre ocupe exactamente longitud
caracteres.
Necesario para que los registros sean de tamaño fijo.
.getBytes("ISO-8859-1"): Convierte el String resultante en un array de bytes usando la codificación ISO-8859-1.
Cada carácter ocupa exactamente 1 byte, lo cual simplifica el cálculo de posiciones en el archivo.
raf.write(byte[]): Escribe esos bytes en el archivo a partir de la posición actual del cursor.
Repite esto para cada campo en orden: número de control, nombre, apellidos, semestre, grupo y carrera.*/
        raf.write(b: ajustar(s: numControl, longitud: NUMCONTROL_LEN).getBytes(charsetName: "ISO-8859-1"));
        raf.write(b: ajustar(s: nombre, longitud: NOMBRE_LEN).getBytes(charsetName: "ISO-8859-1"));
        raf.write(b: ajustar(s: apellidos, longitud: APELLIDOS_LEN).getBytes(charsetName: "ISO-8859-1"));
        raf.write(b: new DecimalFormat(pattern: "00").format(number: semestre).getBytes(charsetName: "ISO-8859-1"));
        raf.write(b: String.valueOf(c: grupo).getBytes(charsetName: "ISO-8859-1"));
        raf.write(b: ajustar(s: carrera, longitud: CARRERA_LEN).getBytes(charsetName: "ISO-8859-1"));
    } catch (IOException ex) {
        Logger.getLogger(name: ArchivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    } //Sin el bloque try/catch, un IOException haría que el método propagara la excepción y posiblemente cerrara la aplicación.
}
```

# OBTENER MATRIZ REGISTRO

```
/*Metodo obtenerMatrizregistro:lee todos los registros válidos guardados en un archivo de acceso aleatorio (RandomAccessFile) y los devuelve con una matriz de cadenas (String[][]).
Cada fila representa un registro, y cada columna un dato: número de control, nombre, apellidos, semestre, grupo y carrera.
*/
public String[][] obtenerMatrizRegistros(int numCol) {
/*public: Puede ser llamado desde otras clases.
String[][]: Devuelve una matriz de cadenas, es decir, una estructura de varias filas y columnas con texto.
int numCol: Indica cuántas columnas debe tener cada fila. (Se espera que sea 6 en este caso, uno por cada campo).*/
    try { //Se usa para manejar errores de lectura de archivo.
        long longitudArchivo = raf.length(); //btiene el tamaño total del archivo en bytes.
//Esto es necesario para saber cuántos registros completos hay en el archivo.
        this.totReg = (int) (longitudArchivo / TAMREG);
/*TAMREG es el tamaño de un registro en bytes (debe ser definido en tu clase).
Divide el tamaño total del archivo entre el tamaño de un registro, lo que da como resultado el número total de registros (totReg).
Se hace un cast a int porque length() devuelve un long.*/
        java.util.ArrayList<String[]> listaRegistros = new java.util.ArrayList<>();
/*Se crea una lista dinámica para almacenar cada registro leído como un arreglo de strings (String[]).
Luego, al final, se convierte en una matriz (String[][])*
        for (int i = 0; i < totReg; i++) {
/*int i = 0      Se declara la variable i e inicia en 0. Representa el índice del registro actual.
i < totReg      Mientras i sea menor que totReg, el ciclo continúa. Es decir, se repite hasta que se hayan procesado todos los registros.
i++            Incrementa i en 1 al final de cada vuelta. Significa: "pasa al siguiente registro".*/
            raf.seek(i * (long) TAMREG);
/* Mueve el cursor al inicio del i-ésimo registro en el archivo.
Multiplica el índice por el tamaño del registro para encontrar la posición correcta en bytes.*/
            byte[] buf = new byte[TAMREG];
//Se lee un registro completo del archivo (que está en formato binario o bytes) y lo guardas en buf.
/*Se crea un buffer de bytes del tamaño de un registro.
readFully(buf) lee los bytes del archivo en esa posición y los guarda en el buffer.*/
            raf.readFully(buf);
```



# OBTENER MATRIZ REGISTRO

```
String numControl = new String(bytes: buf, offset: 0, length: NUMCONTROL_LEN, charsetName: "ISO-8859-1").trim();
// Entonces, esta línea reconstruye el campo número de control desde los bytes leídos en buf.
/*new String Convierte una parte del arreglo de bytes buf en un String de texto legible.
buf      Es el arreglo de bytes que contiene todo el registro leído.
0        Es la posición inicial dentro de buf desde donde quieres empezar a leer. En este caso, desde el inicio.
NUMCONTROL_LEN  Es la cantidad de bytes que vas a leer desde buf para formar el campo numControl.
"ISO-8859-1"    Es el formato de codificación de caracteres. Sirve para interpretar los bytes como texto. Es común en archivos binarios antiguos.
.trim() Elimina espacios en blanco al principio y al final del texto (porque los campos están alineados con espacios)..*/
if (numControl.equals(anObject: "00000000") || numControl.isEmpty()) continue;
/*Esta línea salta registros inválidos o vacíos.
Si el número de control está vacío o es "00000000", simplemente no lo toma en cuenta (pasa al siguiente con continue).*/
String[] fila = new String[numCol];
/*Crea un arreglo de cadenas (fila) que representará un registro completo en forma de texto, con columnas como
si fuera una tabla.*/
fila[0] = numControl;
fila[1] = new String(bytes: buf, offset: NUMCONTROL_LEN, length: NOMBRE_LEN, charsetName: "ISO-8859-1").trim();
/*buf      Arreglo de bytes que tiene todo el registro.
NUMCONTROL_LEN  Posición inicial donde empieza el nombre.
NOMBRE_LEN      Cuántos bytes se usan para el nombre.
new String(..., ..., ..., "ISO-8859-1") Convierte esa parte en texto.
.trim() Elimina espacios de sobra.*/
fila[2] = new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN, length: APELLIDOS_LEN, charsetName: "ISO-8859-1").trim();
/* Empieza después del nombre, y lee lo que corresponde a los apellidos.
(la posición inicial es: lo que ocupan el número de control y el nombre).*/
fila[3] = new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN, length: SEMESTRE_LEN, charsetName: "ISO-8859-1").trim();
/*Lee el campo del semestre, que viene justo después de apellidos.*/
fila[4] = new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN + SEMESTRE_LEN, length: GRUPO_LEN, charsetName: "ISO-8859-1").trim();
/*Lee el campo del grupo, que está después del semestre*/
fila[5] = new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN + SEMESTRE_LEN + GRUPO_LEN, length: CARRERA_LEN, charsetName: "ISO-8859-1").trim();
/*Lee el campo de la carrera, que viene al final.*/
listaRegistros.add(e: fila); //Agrega la fila (registro ya convertido a texto y separado por campos) a la lista de registros leídos.
}
```



# OBTENER MATRIZ REGISTRO/BUSCAR DATO

```
        listaRegistros.add(e: fila);//Agrega la fila (registro ya convertido a texto y separado por campos) a la lista de registros
    }

    String[][] registros = new String[listaRegistros.size()][numCol];
    /*Se convierte la ArrayList en una matriz (String[][]), porque eso es más fácil de usar en interfaces gráficas o tablas.
    Devuelve la matriz completa con todos los registros válidos.*/
    return listaRegistros.toArray(a: registros);
} catch (IOException ex) {
    Logger.getLogger(archivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
    return null;//esto es por si ocurre un error lo atrapa y hace que no se cierre abruptamente
}
}

public String[] buscarDato(String nc) {
    try {
        long pos = establecerPosicionByte(numeroControl: nc);
        // calcula la posición en el archivo donde se encuentra el registro con número de control nc.
        raf.seek(pos);//Mueve el "cursor" del archivo (RandomAccessFile) a esa posición para poder leer el registro.
        byte[] buf = new byte[TAMREG];
        /*Se crea un arreglo de bytes del tamaño del registro (TAMREG).
        Se leen todos los bytes del registro completo y se almacenan en buf.*/
        raf.readFully(b: buf);
        String nControl = new String(bytes: buf, offset: 0, length: NUMCONTROL_LEN, charsetName: "ISO-8859-1").trim();
        /*Se convierte la primera parte de buf (desde el byte 0 hasta NUMCONTROL_LEN) en texto.
        .trim() elimina espacios en blanco sobrantes.*/
        if (!nControl.equals(anObject: nc) || nControl.equals(anObject: "00000000")) return null;
        /*Si el número de control no es igual al que buscabas (nc), o si está vacío ("00000000"),
        se descarta el registro y retorna null.*/
        return new String[]{
            nControl,
            new String(bytes: buf, offset: NUMCONTROL_LEN, length: NOMBRE_LEN, charsetName: "ISO-8859-1").trim(),
        }
    }
}
```

# BUSCAR DATO/ELIMINAR LINEA

```
/*Fila  Campo extraído  Desde qué posición en buf      Cuántos bytes  Qué representa
[0]     nControl        0          NUMCONTROL_LEN  Número de control
[1]     nombre  NUMCONTROL_LEN  NOMBRE_LEN        Nombre del alumno
[2]     apellidos      NUMCONTROL_LEN + NOMBRE_LEN  APELLIDOS_LEN  Apellidos
[3]     semestre      Suma anterior + APELLIDOS_LEN  SEMESTRE_LEN  Semestre
[4]     grupo    Suma anterior + SEMESTRE_LEN  GRUPO_LEN      Grupo
[5]     carrera Suma anterior + GRUPO_LEN      CARRERA_LEN      Carrera o programa*/

        new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN, length: APELLIDOS_LEN, charsetName: "ISO-8859-1").trim(),
        new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN, length: SEMESTRE_LEN, charsetName: "ISO-8859-1").trim(),
        new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN + SEMESTRE_LEN, length: GRUPO_LEN, charsetName: "ISO-8859-1").trim(),
        new String(bytes: buf, NUMCONTROL_LEN + NOMBRE_LEN + APELLIDOS_LEN + SEMESTRE_LEN + GRUPO_LEN, length: CARRERA_LEN, charsetName: "ISO-8859-1").trim()
    };
} catch (IOException ex) {
    Logger.getLogger(ArchivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    return null;
} //Si hay una excepción (por ejemplo, se intenta leer fuera del archivo), se registra el error y el método devuelve null.
}

public void eliminarLinea(String numControl) {
    try { //Todas las operaciones de E/S (entrada/salida) en Java pueden lanzar excepciones, como cuando el disco está lleno o el archivo está corrupto.
        //Calcula en qué parte del archivo se debe escribir el registro.
        long pos = establecerPosicionByte(numeroControl: numControl);
        //Se calcula en qué posición del archivo está el registro que tiene el número de control que se quiere eliminar.
        raf.seek(pos); //Mueve el puntero del archivo (RandomAccessFile) exactamente al inicio de ese registro.
        raf.write(b: ajustar(s: "00000000", longitud: NUMCONTROL_LEN).getBytes(charsetName: "ISO-8859-1"));
    } catch (IOException ex) {
        Logger.getLogger(ArchivoOrgnDir.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        return null;
    }
}

/*ajustar("00000000", NUMCONTROL_LEN):
Asegura que la cadena "00000000" tenga el mismo tamaño en caracteres que el campo del número de control (NUMCONTROL_LEN).
Si el campo es más largo que 8 caracteres, la función lo rellena con espacios o ceros (dependiendo de cómo esté escrita ajustar()).
.getBytes("ISO-8859-1"):
Convierte la cadena "00000000" a un arreglo de bytes en el formato adecuado del archivo.
raf.write(...):
Escribe solo esa parte del campo del número de control en el archivo.
Así el resto del registro (nombre, apellidos, etc.) queda intacto, pero ya no se reconocerá este registro como válido.*/
```



# CREAR LINEA/ELIMINAR ARCHIVO

```
    } catch (IOException ex) {
        Logger.getLogger(ArchivoOrgnDir.class.getName()).log(Level.SEVERE, msg:null, thrown: ex);
    }
}

public void crearLinea(String linea) {
    /*Este método convierte una línea de texto en un registro estructurado
y llama a otro método para guardarlo en el archivo binario.*/
    String[] datos = linea.split(regex: ",");//Toma la cadena de entrada linea y la divide por comas.
    if (datos.length == 6) {
        /*Verifica que se hayan capturado exactamente 6 partes
(estó es importante para evitar errores si el formato está mal).
Si no hay 6 partes, no hace nada.*/
        escribirRegistro(datos[0], datos[1], datos[2], semestre: Integer.parseInt(datos[3]), grupo: datos[4].charAt(index: 0), datos[5]);
        //Si los datos son correctos, llama al método escribirRegistro con los valores convertidos al tipo adecuado.
    }
}

public void eliminarArchivo(String archivoEliminar) {
    /*elimina un archivo del sistema de archivos. El archivo que se elimina es el que se pasa como
parámetro en archivoEliminar, pero antes verifica si puede establecer el flujo hacia ese archivo
(es decir, si el archivo existe y se puede abrir correctamente).*/
    if (this.establecerFlujo(nombreArchivo: archivoEliminar)) {
        //Si se puede establecer el flujo (es decir, el archivo existe y se puede trabajar con él), se ejecuta lo siguiente.
        fichero.delete();
        //Elimina físicamente el archivo del disco usando el método delete() de la clase File.
        //Si el archivo fue encontrado y no está siendo usado por otro proceso, será eliminado.
    }
}
```



# RENOMBRAR ARCHIVO/VALIDA CONTROL

```
public void renombrarArchivo(String archivoRenombrar, String nuevoNombre) {  
    //Si el archivo existe, guarda la referencia en la variable this.fichero (de tipo File) y devuelve true.  
    //Solo si se puede acceder correctamente al archivo original, continúa con el cambio de nombre.  
    if (this.establecerFlujo(nombreArchivo: archivoRenombrar)) {  
        File nuevo = new File(pathname: nuevoNombre);  
        //Crea un nuevo objeto de tipo File que representa el nuevo nombre del archivo.  
        this.fichero.renameTo(dest: nuevo);  
        //Aquí se intenta renombrar o mover el archivo actual (fichero) al nuevo nombre.  
    }  
}  
  
public boolean validaControl(String numControl) {  
    /*ste método valida que el número de control (numControl) cumpla con un formato específico:  
    Debe contener exactamente 8 dígitos numéricos (del 0 al 9).  
    Si lo cumple, devuelve true.  
    Si no lo cumple, devuelve false.  
    se usa para verificar si una cadena de texto (String) coincide exactamente  
    con un patrón especificado mediante una expresión regular (regex).*/  
    return numControl.matches(regex: "[0-9]{8}$");  
}  
}
```

# MÉTODOS CLAVE:

- `abrirArchivoRAF(nombreArchivo):`
- Abre o crea un archivo con acceso de lectura y escritura. Usa `RandomAccessFile`.
- `cerrarArchivo():`
- Cierra el archivo si está abierto, liberando recursos.
- `escribirRegistro(...):`
- Escribe un registro completo en la posición correspondiente del archivo. Los datos se ajustan a longitudes fijas y se escriben como bytes usando codificación "ISO-8859-1".
- `obtenerMatrizRegistros(numCol):`
- Lee todos los registros válidos del archivo y los devuelve como una matriz de cadenas (`String[][]`). Ignora los registros vacíos o inválidos (por ejemplo, los que tienen número de control "00000000").

# VISTACRUDESTUDIANTES

Objetivo:

Actuar como la interfaz gráfica principal del sistema de gestión de estudiantes, permitiendo al usuario realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) a través de botones e interacción con una tabla visual (JTable). Forma parte de la arquitectura MVC como la Vista, y se comunica con un controlador (ControladorCRUDEstudiantes) y un modelo (ArchivoOrgnDir) para gestionar los datos almacenados en un archivo binario.





# VISTA CRUDE ESTUDIANTES

```
package vista;
// Define que esta clase pertenece al paquete 'vista', el cual normalmente contiene las interfaces gráficas (Vistas) en un patrón MVC.
import controlador.ControladorCRUDEstudiantes;
// Importa la clase ControladorCRUDEstudiantes que gestiona la lógica entre la vista y el modelo.
import java.awt.Color;
// Importa la clase Color para usar colores personalizados en la interfaz (por ejemplo, para resaltar una fila)
import java.awt.event.KeyEvent;
// Importa la clase KeyEvent que permite manejar eventos del teclado (por ejemplo, si el usuario presiona una tecla).
import javax.swing.JOptionPane;
// Importa la clase JOptionPane que permite mostrar cuadros de diálogo como mensajes, confirmaciones o alertas.
import javax.swing.table.DefaultTableModel;
// Importa el modelo de tabla que se usa para manejar los datos que se muestran en un JTable.
import modelo.ArchivoOrgnDir;
// Importa la clase ArchivoOrgnDir que pertenece al modelo y maneja la lectura y escritura en el archivo.
public class VistaCRUDEstudiantes extends javax.swing.JFrame {
// Esta clase define una ventana gráfica (hereda de JFrame) que se usará como interfaz principal del CRUD.
    ControladorCRUDEstudiantes objControladorCRUDEst;
// Declara un objeto del controlador que conectará esta vista con el modelo.
    ArchivoOrgnDir objArchivo; // Pertenece a modelo
// Declara un objeto del modelo que maneja directamente los datos del archivo.
    private int filaResaltada = -1; // -1 significa que no hay fila resaltada
// Esta variable guarda el índice de la fila resaltada en la tabla. Se usa para darle color a una fila si es necesario.
    public VistaCRUDEstudiantes() {
        // Constructor de la clase, se ejecuta al crear una nueva ventana VistaCRUDEstudiantes.
        initComponents();
        // Método generado por NetBeans (o cualquier GUI Builder) que inicializa todos los componentes gráficos (botones, cajas de texto, tabla,
        this.objArchivo = new ArchivoOrgnDir();
        // Crea una nueva instancia del modelo, que permite trabajar con el archivo de estudiantes.
        this.objControladorCRUDEst = new ControladorCRUDEstudiantes(vista: this, archivo: this.objArchivo);
        // Crea una nueva instancia del controlador, pasándole como parámetros esta vista (para que pueda acceder a los componentes) y el modelo
        this.objControladorCRUDEst.llenarTabla();
        // Llama al método llenarTabla del controlador para que la tabla de estudiantes se llene con los datos leídos del archivo.
    }
}
```

# VISTA CRUDE ESTUDIANTES

@SuppressWarnings("unchecked")

Generated Code

```
private void btnAgregarMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    FrmAgregarEstudiante objAgregarEstudiante = new FrmAgregarEstudiante(objCtrlCRUDEst:this.objControladorCRUDEst);  
    objAgregarEstudiante.setVisible(b: true);  
}  
  
private void btnEliminarMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    FrmEliminarEstudiante objEliminarEstudiante = new FrmEliminarEstudiante(objCtrlCRUDEst:this.objControladorCRUDEst);  
    objEliminarEstudiante.setVisible(b: true);  
}  
  
private void btnEditarMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    FrmModificarEstudiante objModificarEstudiante = new FrmModificarEstudiante(controlador:this.objControladorCRUDEst);  
    objModificarEstudiante.setVisible(b: true);  
}
```



# BTN BUSCAR ACTION PERFORMED

```
private void btnBuscarActionPerformed(java.awt.event.ActionEvent evt) {  
    // Este método se ejecuta automáticamente cuando el usuario hace clic en el botón "Buscar".  
    String numControlBuscar = txtNumControlBuscar.getText().trim();  
    // Se obtiene el texto escrito en la caja de texto para buscar por número de control.  
    // .trim() elimina espacios al inicio y al final.  
    if (!numControlBuscar.isEmpty()) {  
        // Verifica que el campo no esté vacío  
        String[] datosEstudiante = objControladorCRUDEst.buscarRegistro(nc: numControlBuscar);  
        // Se llama al controlador para buscar el registro en el archivo usando el número de control.  
        // Si se encuentra, se regresa un arreglo con los datos del estudiante; si no, regresa null.  
        if (datosEstudiante != null) {  
            // Si se encontró un registro en el archivo:  
            DefaultTableModel model = (DefaultTableModel) jTableEstudiantes.getModel();  
            // Se obtiene el modelo de la tabla para poder acceder a sus filas y columnas.  
            boolean encontrado = false;  
            // Variable para saber si el número de control también fue encontrado en la tabla visual.  
            for (int i = 0; i < model.getRowCount(); i++) {  
                // Se recorre cada fila de la tabla  
                Object valorCelda = model.getValueAt(row:i, column: 0);  
                // Se obtiene el valor de la primera columna (número de control) de la fila actual.  
                if (valorCelda != null) {  
                    String numControl = valorCelda.toString().trim();  
                    // Se convierte a texto y se eliminan espacios.  
                    if (numControl.equals(anObject: numControlBuscar)) {  
                        // Si el número de control de la fila coincide con el que se está buscando:  
                        jTableEstudiantes.setRowSelectionInterval(index0: i, index1: i);  
                        // Selecciona visualmente esa fila en la tabla.  
                        jTableEstudiantes.scrollRectToVisible(aRect: jTableEstudiantes.getCellRect(row:i, column: 0, includeSpacing:true));  
                        // Desplaza la vista de la tabla para que la fila encontrada sea visible.  
                        filaResaltada = i;  
                        // Guarda el índice de la fila resaltada (posiblemente para cambiarle el color después).  
                        jTableEstudiantes.repaint();  
                    }  
                }  
            }  
        }  
    }  
}
```



# BTN BUSCAR ACTION PERFORMED

```
// Guarda el índice de la fila resaltada (posiblemente para cambiarle el color despues).
jtblEstudiantes.repaint();
// Vuelve a dibujar la tabla (útil si hay un render especial para la fila resaltada).
encontrado = true;
break; // Sale del ciclo porque ya encontró la fila.
    }
}
}
if (!encontrado) {
    // Si el número de control existía en el archivo, pero no se encontró en la tabla visual:
    JOptionPane.showMessageDialog(parentComponent: this, message: "Estudiante no encontrado en la tabla.", title: "Información", messageType: JOptionPane.INFORMATION_MESSAGE);
}
} else {
    // Si no se encontró el número de control en el archivo (datosEstudiante es null):
    JOptionPane.showMessageDialog(parentComponent: this, message: "Estudiante no encontrado en el archivo.", title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
}
} else {
    // Si el usuario no ingresó ningún número de control para buscar:
    JOptionPane.showMessageDialog(parentComponent: this, message: "Por favor, ingresa un número de control para buscar.", title: "Advertencia", messageType: JOptionPane.WARNING_MESSAGE);
}
txtNumControlBuscar.setText(t: "");
// Limpia la caja de texto después de buscar.
}

public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new VistaCRUDEstudiantes().setVisible(true);
        }
    });
}
```

# VISTA CRUDE ESTUDIANTES

```
public static void main(String args[]) {  
    /* Set the Nimbus look and feel */  
    Look and feel setting code (optional)  
  
    /* Create and display the form */  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new VistaCRUDEstudiantes().setVisible(b: true);  
        }  
    });  
}
```

```
// Variables declaration - do not modify  
private javax.swing.JButton btnAgregar;  
private javax.swing.JButton btnBuscar;  
private javax.swing.JButton btnEditar;  
private javax.swing.JButton btnEliminar;  
private javax.swing.JLabel jLabel1;  
private javax.swing.JLabel jLabel2;  
private javax.swing.JScrollPane jScrollPane1;  
public javax.swing.JTable jtblEstudiantes;  
private javax.swing.JTextField txtNumControlBuscar;  
// End of variables declaration
```

```
}
```

# FRM AGREGAR ESTUDIANTE

Objetivo:

La clase FrmAgregarEstudiante sirve como vista en el patrón MVC (Modelo-Vista-Controlador), encargándose de que el usuario pueda ingresar y enviar información de nuevos estudiantes a través de una ventana amigable.



The illustration shows a blue computer monitor with a white form titled "Tecnológico del Valle de Oaxaca" and "Registrar Estudiantes". The form contains several input fields: "Num Control", "Nombre", "Apellidos", "Semestre" (a dropdown menu with "1" selected), "Grupo" (radio buttons for A, B, and C, with A selected), and "Carrera" (a dropdown menu with "Ing. Informática" selected). At the bottom of the form is a green button with a white downward arrow icon and the text "Guardar". The monitor has a black bezel with a camera icon and a lock icon on the bottom right. A black mouse cursor is pointing at the top right corner of the monitor.

Tecnológico del Valle de Oaxaca  
Registrar Estudiantes

Num Control

Nombre

Apellidos

Semestre

Grupo ☒ A ☐ B ☐ C

Carrera

 Guardar



# FRM AGREGAR ESTUDIANTE

```
package vista;
// Esta línea indica que esta clase pertenece al paquete "vista", es decir, a la parte de la interfaz gráfica (GUI) de la aplicación.
import controlador.ControladorCRUDEstudiantes;
// Importa la clase del controlador que maneja la lógica para CRUD (Crear, Leer, Actualizar y Eliminar) estudiantes.
import java.awt.event.KeyEvent;
// Importa la clase KeyEvent, que se usa si quieres detectar cuando el usuario presiona teclas en los componentes (aunque en este fragmento no s
import javax.swing.JOptionPane;
// Importa la clase JOptionPane para mostrar cuadros de diálogo (mensajes emergentes al usuario).
public class FrmAgregarEstudiante extends javax.swing.JFrame {
    // Define una clase pública llamada FrmAgregarEstudiante que hereda de JFrame, lo que significa que esta clase representa una ventana gráfic
    ControladorCRUDEstudiantes objControladorCRUDEst;
    // Se declara un objeto del tipo ControladorCRUDEstudiantes. Este será el controlador que conecta la vista con el modelo (la lógica del prog
    public FrmAgregarEstudiante(ControladorCRUDEstudiantes objCtrlCRUDEst) {
        // Constructor de la clase. Recibe un objeto del tipo ControladorCRUDEstudiantes como parámetro.
        initComponents();
        // Método generado automáticamente que inicializa todos los componentes gráficos del formulario
        // (botones, etiquetas, campos de texto, etc.).
        this.objControladorCRUDEst = objCtrlCRUDEst;
        // Asigna el objeto recibido como parámetro al atributo de la clase.
        // Esto permite que la ventana pueda usar el controlador para guardar, buscar o eliminar estudiantes.
    }

    private void limpiarCampos() {
        txtNumControl.setText(t: "");
        txtNombre.setText(t: "");
        txtApellidos.setText(t: "");
        cboSemestre.setSelectedIndex(anIndex: 0);
        cboCarrera.setSelectedIndex(anIndex: 0);
        rdbtnGrupoA.setSelected(b: true); // Seleccionar el grupo A por defecto
        txtNumControl.requestFocus(); // Regresar el foco al primer campo
    }
}
```

# FRM AGREGAR ESTUDIANTE

@SuppressWarnings("unchecked")

Generated Code

```
private void txtNumControlKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.txtNombre.requestFocus();  
}
```

```
private void txtNombreKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.txtApellidos.requestFocus();  
}
```

```
private void txtApellidosKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.cboSemestre.requestFocus();  
}
```

```
private void cboSemestreKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.rdbtnGrupoA.requestFocus();  
}
```

```
private void rdbtnGrupoAKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.cboCarrera.requestFocus();  
}
```

```
private void rdbtnGrupoBKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.cboCarrera.requestFocus();  
}
```

```
private void rdbtnGrupoCKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.cboCarrera.requestFocus();  
}
```

```
private void cboCarreraKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)  
        this.btnGuardar.requestFocus();  
}
```

```
private void btnGuardarKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    this.agregarEstudiante();  
}
```

```
private void btnGuardarMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    this.agregarEstudiante();  
}
```



```

private void agregarEstudiante() {
// Este método se llama cuando el usuario desea agregar un nuevo estudiante.
// Obtener los datos del formulario
String numControl = this.txtNumControl.getText();
// Se obtiene el número de control desde la caja de texto correspondiente.
String nombre = this.txtNombre.getText();
// Se obtiene el nombre ingresado por el usuario.
String ape = this.txtApellidos.getText();
// Se obtienen los apellidos del estudiante.
int semestre = Integer.parseInt(s: this.cboSemestre.getSelectedItem().toString());
// Se obtiene el semestre seleccionado en el comboBox (desplegable),
// se convierte a cadena y luego a número entero.
char grupo;
if (this.rdbtnGrupoA.isSelected()) grupo = 'A';
else if (this.rdbtnGrupoB.isSelected()) grupo = 'B';
else grupo = 'C';
// Se determina cuál de los botones de grupo (A, B o C) está seleccionado
// y se asigna su valor como un carácter.
String carrera = this.cboCarrera.getSelectedItem().toString();
// Se obtiene la carrera seleccionada del comboBox correspondiente.
// Validar el número de control
if (this.objControladorCRUDEst.validaNumControl(numControl)) {
// Se llama al método del controlador para validar si el número de control es correcto.
// Si es válido:
JOptionPane.showMessageDialog(parentComponent: this, message: "Control Válido");
// Muestra un mensaje indicando que el número de control es válido.
// Guardar el estudiante usando acceso directo
this.objControladorCRUDEst.guardarRegistro(nc: numControl, nom: nombre, ape, sem: semestre, gpo: grupo, carrera);
// Se llama al método guardarRegistro para almacenar la información del nuevo estudiante.
JOptionPane.showMessageDialog(parentComponent: this, message: "Registro guardado con éxito");
// Muestra un mensaje indicando que el registro fue exitoso.
}
}

```



```

JOptionPane.showMessageDialog(parentComponent: this, message: "Registro guardado con éxito");
// Muestra un mensaje indicando que el registro fue exitoso.
// Limpiar los campos del formulario
limpiarCampos();
// Limpia todos los campos del formulario para permitir un nuevo ingreso.
} else {
    // Si el número de control no es válido:
    JOptionPane.showMessageDialog(parentComponent: this, message: "Control no Válido");
    // Muestra un mensaje de advertencia.
    this.txtNumControl.requestFocus();
    // Coloca el cursor nuevamente en el campo de número de control para que el usuario lo
}
}

```

```

// Variables declaration - do not modify
private javax.swing.JButton btnGuardar;
private javax.swing.JComboBox<String> cboCarrera;
private javax.swing.JComboBox<String> cboSemestre;
private javax.swing.ButtonGroup gpoBotonesGrupo;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JRadioButton rdbtnGrupoA;
private javax.swing.JRadioButton rdbtnGrupoB;
private javax.swing.JRadioButton rdbtnGrupoC;
private javax.swing.JTextField txtApellidos;
private javax.swing.JTextField txtNombre;
private javax.swing.JTextField txtNumControl;

```

# FRM ELIMINAR ESTUDIANTE

Objetivo:

- Buscar un estudiante por número de control usando el botón de búsqueda o presionando Enter.
- Mostrar sus datos en los campos correspondientes si se encuentra.
- Eliminar el registro si el usuario lo confirma.
- Deshabilitar campos y limpiar la interfaz tras eliminar.
- Habilitar/Deshabilitar componentes según el flujo.



The screenshot shows a web application window with a purple border and a light blue background. The title bar at the top has a close button (X) and a minus sign (-). A black mouse cursor is pointing at the close button. The main content area is titled "Tecnológico del Valle de Oaxaca" and "Eliminar Estudiante". It contains several input fields: "Num Control" (with a search icon), "Nombre", "Apellidos", "Semestre" (a dropdown menu showing "1"), "Grupo" (radio buttons for A, B, and C, with A selected), and "Carrera" (a dropdown menu showing "Ing. Informática"). At the bottom, there is a red circular button with a trash icon and the text "Eliminar". The bottom of the window has a purple bar with three icons: a hamburger menu, a camera, and a lock.

# FRM ELIMINAR ESTUDIANTE

```
package vista; // Define que esta clase pertenece al paquete 'vista', es decir, la parte visual de la aplicación (interfaz gráfica).
import controlador.ControladorCRUDEstudiantes; // Importa la clase del controlador que maneja la lógica de negocio para CRUD de estudiantes.
import java.awt.event.KeyEvent; // Importa la clase necesaria para manejar eventos de teclado (aunque aquí no se usa directamente).
import javax.swing.JOptionPane; // Importa la clase que permite mostrar cuadros de diálogo como mensajes, advertencias y confirmaciones.
public class FrmEliminarEstudiante extends javax.swing.JFrame { // Declara la clase FrmEliminarEstudiante, que hereda de JFrame (una ventana de
    ControladorCRUDEstudiantes objControladorCRUDEst; // Declara un objeto del controlador para manejar las operaciones con estudiantes.
    public FrmEliminarEstudiante(ControladorCRUDEstudiantes objCtrlCRUDEst) { // Constructor de la ventana FrmEliminarEstudiante, recibe el cont
        initComponents(); // Llama al método que construye todos los componentes gráficos de la interfaz (botones, etiquetas, campos, etc.).
        this.objControladorCRUDEst = objCtrlCRUDEst; // Asigna el controlador recibido al atributo de esta clase para usarlo internamente.
        this.des_habilitaComponentes(valor: false); // Llama a un método que probablemente desactiva ciertos componentes gráficos al iniciar la ventan
    }
    //Habilitar y deshabilitar los componentes
    private void des_habilitaComponentes(boolean valor){
        this.txtNombre.setEnabled(enabled:valor);
        this.txtApellidos.setEnabled(enabled:valor);
        this.cboSemestre.setEnabled(b: valor);
        this.rdbtnGrupoA.setEnabled(b: valor);
        this.rdbtnGrupoB.setEnabled(b: valor);
        this.rdbtnGrupoC.setEnabled(b: valor);
        this.cboCarrera.setEnabled(b: valor);
        this.btnEliminar.setEnabled(b: valor);
    }
    private void limpiarCampos() {
        txtNumControl.setText(t: "");
        txtNombre.setText(t: "");
        txtApellidos.setText(t: "");
        cboSemestre.setSelectedIndex(anIndex: 0);
        cboCarrera.setSelectedIndex(anIndex: 0);
        rdbtnGrupoA.setSelected(b: true); // Seleccionar el grupo A por defecto
        txtNumControl.requestFocus(); // Regresar el foco al primer campo
    }
}
```



# FRM ELIMINAR ESTUDIANTE

```
private void btnEliminarKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:

    if (evt.getKeyCode() == KeyEvent.VK_ENTER)
        this.eliminar();
    limpiarCampos();
}

private void btnBuscarKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if (evt.getKeyCode() == KeyEvent.VK_ENTER)
        this.buscar();
}

private void txtNumControlKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if (evt.getKeyCode() == KeyEvent.VK_ENTER)
        this.buscar();
    if (evt.getKeyCode() == KeyEvent.VK_ENTER)
        this.btnEliminar.requestFocus();
}

private void btnBuscarMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    this.buscar();
}

private void btnEliminarMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    this.eliminar();
    limpiarCampos();
}
```

```
private void limpiarSeleccionGrupo() {
    // Limpiar todos los botones de grupo (asegurándonos que ninguno esté seleccionado)
    rdbtnGrupoA.setSelected(b: false);
    rdbtnGrupoB.setSelected(b: false);
    rdbtnGrupoC.setSelected(b: false);
}

private void buscar() {
    String numControl = this.txtNumControl.getText();
    String registro[] = this.objControladorCRUDEst.buscarRegistro(nc: numControl);
    if (registro != null) {
        this.txtNombre.setText(registro[1]);
        this.txtApellidos.setText(registro[2]);
        this.cboSemestre.setSelectedItem(anObject: String.valueOf(i: Integer.parseInt(registro[3])));
        this.rdbtnGrupoA.setSelected(b: false);
        this.rdbtnGrupoB.setSelected(b: false);
        this.rdbtnGrupoC.setSelected(b: false);
        switch (registro[4]) {
            case "A":
                this.rdbtnGrupoA.setSelected(b: true);
                break;
            case "B":
                this.rdbtnGrupoB.setSelected(b: true);
                break;
            case "C":
                this.rdbtnGrupoC.setSelected(b: true);
                break;
        }
        this.cboCarrera.setSelectedItem(registro[5]);
        this.btnEliminar.setEnabled(b: true);
    } else {
        JOptionPane.showMessageDialog(parentComponent: this, message: "El registro no Existe");
    }
}
```

# FRM ELIMINAR ESTUDIANTE

```
private void eliminar() {
    int respuesta = JOptionPane.showConfirmDialog(parentComponent: this, message: "Estas seguro de eliminar el registro");
    if(respuesta==0){
        String numControl = this.txtNumControl.getText();
        this.objControladorCRUDEst.eliminarRegistro(nc: numControl);
        JOptionPane.showMessageDialog(parentComponent: this.objControladorCRUDEst.objVistaCRUDEst, message: "Registro Eliminado");
        this.btnEliminar.setEnabled(b: false);
        this.txtNumControl.setText(t: "");
        this.txtNumControl.requestFocus();
    }
}

// Variables declaration - do not modify
private javax.swing.JButton btnBuscar;
private javax.swing.JButton btnEliminar;
private javax.swing.JComboBox<String> cboCarrera;
private javax.swing.JComboBox<String> cboSemestre;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JRadioButton rdbtnGrupoA;
private javax.swing.JRadioButton rdbtnGrupoB;
private javax.swing.JRadioButton rdbtnGrupoC;
private javax.swing.JTextField txtApellidos;
private javax.swing.JTextField txtNombre;
private javax.swing.JTextField txtNumControl;
// End of variables declaration
}
```

# FRM MODIFICAR ESTUDIANTE

Objetivo:

Proporcionar una interfaz gráfica que permita al usuario buscar un estudiante registrado mediante su número de control, visualizar sus datos actuales y modificar su información de manera sencilla y precisa. Esta clase facilita la edición de los registros almacenados en el archivo con organización por dirección, asegurando que los cambios se guarden correctamente y manteniendo la integridad de los datos.



The screenshot shows a web browser window with a yellow header and a light blue content area. The title is 'Tecnológico del Valle de Oaxaca' and the subtitle is 'Modificar Estudiante'. The form contains several input fields: 'Num Control' (text), 'Nombre' (text), 'Apellidos' (text), 'Semestre' (dropdown menu with '1' selected), 'Grupo' (radio buttons for A, B, and C, with A selected), and 'Carrera' (dropdown menu with 'Ing. Informática' selected). There is a magnifying glass icon in the top right corner of the form area. At the bottom of the form is a button labeled 'Actualizar' with a small icon of a person. The browser window has a yellow border and a black cursor pointing at the top right corner.

Tecnológico del Valle de Oaxaca

Modificar Estudiante

Num Control

Nombre

Apellidos

Semestre

Grupo ☒ A ☐ B ☐ C

Carrera

 Actualizar



# FRM MODIFICAR ESTUDIANTE

```
package vista;
// Define que esta clase forma parte del paquete 'vista', es decir, la interfaz gráfica del sistema.
import controlador.ControladorCRUDEstudiantes;
// Importa la clase del controlador que gestiona la lógica del CRUD de estudiantes.
import java.awt.event.KeyEvent;
// Importa la clase para manejar eventos de teclado (aunque no se usa en este fragmento).
import javax.swing.JFrame;
// Importa la clase JFrame, que permite crear ventanas en una aplicación gráfica con Swing.
import javax.swing.JOptionPane;
// Importa la clase JOptionPane, utilizada para mostrar mensajes emergentes como alertas, advertencias, etc.
import modelo.ArchivoOrgnDir;
// Importa la clase que maneja el acceso a archivos (probablemente RandomAccessFile para datos de estudiantes).
public class FrmModificarEstudiante extends JFrame {
// Declara la clase FrmModificarEstudiante que hereda de JFrame, por lo tanto representa una ventana para modificar estudiantes.
    private ControladorCRUDEstudiantes objControladorCRUDEst;
    // Declaración del atributo que representa al controlador del CRUD, encargado de comunicarse con el modelo.
    private ArchivoOrgnDir archivo;
    // Declaración del atributo que representa el manejador del archivo donde se guardan los datos de estudiantes.
    public FrmModificarEstudiante(ControladorCRUDEstudiantes controlador) {
    // Constructor de la clase, que recibe un objeto controlador para manipular datos de estudiantes.
        initComponents();
        // Llama al método generado automáticamente por NetBeans o cualquier IDE con diseñador gráfico,
        // encargado de crear y posicionar los componentes visuales de la ventana.
        this.objControladorCRUDEst = controlador;
        // Asigna el controlador recibido como parámetro al atributo de la clase, para que se pueda usar internamente.
        this.archivo = new ArchivoOrgnDir();
        // Crea una nueva instancia del archivo para realizar operaciones (como buscar o editar registros).
    }
```

# FRM MODIFICAR ESTUDIANTE

```
package vista;
// Define que esta clase forma parte del paquete 'vista', es decir, la interfaz gráfica del sistema.
import controlador.ControladorCRUDEstudiantes;
// Importa la clase del controlador que gestiona la lógica del CRUD de estudiantes.
import java.awt.event.KeyEvent;
// Importa la clase para manejar eventos de teclado (aunque no se usa en este fragmento).
import javax.swing.JFrame;
// Importa la clase JFrame, que permite crear ventanas en una aplicación gráfica con Swing.
import javax.swing.JOptionPane;
// Importa la clase JOptionPane, utilizada para mostrar mensajes emergentes como alertas, advertencias, etc.
import modelo.ArchivoOrgnDir;
// Importa la clase que maneja el acceso a archivos (probablemente RandomAccessFile para datos de estudiantes).
public class FrmModificarEstudiante extends JFrame {
// Declara la clase FrmModificarEstudiante que hereda de JFrame, por lo tanto representa una ventana para modificar estudiantes.
    private ControladorCRUDEstudiantes objControladorCRUDEst;
    // Declaración del atributo que representa al controlador del CRUD, encargado de comunicarse con el modelo.
    private ArchivoOrgnDir archivo;
    // Declaración del atributo que representa el manejador del archivo donde se guardan los datos de estudiantes.
    public FrmModificarEstudiante(ControladorCRUDEstudiantes controlador) {
    // Constructor de la clase, que recibe un objeto controlador para manipular datos de estudiantes.
        initComponents();
        // Llama al método generado automáticamente por NetBeans o cualquier IDE con diseñador gráfico,
        // encargado de crear y posicionar los componentes visuales de la ventana.
        this.objControladorCRUDEst = controlador;
        // Asigna el controlador recibido como parámetro al atributo de la clase, para que se pueda usar internamente.
        this.archivo = new ArchivoOrgnDir();
        // Crea una nueva instancia del archivo para realizar operaciones (como buscar o editar registros).
    }
```

# FRM MODIFICAR ESTUDIANTE

```
private void limpiarCampos() {  
    txtNumControl.setText(t: "");  
    txtNombre.setText(t: "");  
    txtApellidos.setText(t: "");  
    cboSemestre.setSelectedIndex(anIndex: 0);  
    cboCarrera.setSelectedIndex(anIndex: 0);  
    rdbtnGrupoA.setSelected(b: true); // Seleccionar el grupo A por defecto  
    txtNumControl.requestFocus(); // Regresar el foco al primer campo  
}
```

```
private void cboSemestreKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyCode() == KeyEvent.VK_ENTER)  
        this.rdbtnGrupoA.requestFocus();  
}
```



# FRM MODIFICAR ESTUDIANTE

```
private void btnActualizarMouseClicked(java.awt.event.MouseEvent evt) {  
    String numControl = txtNumControl.getText();  
    String nuevoNombre = txtNombre.getText();  
    String nuevoApellido = txtApellidos.getText();  
  
    // Convertir el semestre a int  
    int nuevoSemestre = Integer.parseInt(s: cboSemestre.getSelectedItem().toString());  
  
    // Obtener el grupo como char  
    char nuevoGrupo = ' ';  
    if (rdbtnGrupoA.isSelected()) {  
        nuevoGrupo = 'A';  
    } else if (rdbtnGrupoB.isSelected()) {  
        nuevoGrupo = 'B';  
    } else if (rdbtnGrupoC.isSelected()) {  
        nuevoGrupo = 'C';  
    }  
  
    String nuevaCarrera = cboCarrera.getSelectedItem().toString();  
  
    // Llamar al controlador con los tipos correctos  
    objControladorCRUDEst.editarRegistro(nc: numControl, nuevoNom: nuevoNombre, nuevoApe: nuevoApellido, nuevoSem: nuevoSemestre, nuevoGpo: nuevoGrupo, nue  
  
    limpiarCampos();  
}
```

# FRM MODIFICAR ESTUDIANTE

```
private void btnActualizarKeyPressed(java.awt.event.KeyEvent evt) {  
    String numControl = txtNumControl.getText();  
    String nuevoNombre = txtNombre.getText();  
    String nuevoApellido = txtApellidos.getText();  
  
    // Convertir el semestre a int  
    int nuevoSemestre = Integer.parseInt(s: cboSemestre.getSelectedItem().toString());  
  
    // Obtener el grupo como char  
    char nuevoGrupo = ' ';  
    if (rdbtnGrupoA.isSelected()) {  
        nuevoGrupo = 'A';  
    } else if (rdbtnGrupoB.isSelected()) {  
        nuevoGrupo = 'B';  
    } else if (rdbtnGrupoC.isSelected()) {  
        nuevoGrupo = 'C';  
    }  
  
    String nuevaCarrera = cboCarrera.getSelectedItem().toString();  
  
    // Llamar al controlador con los tipos correctos  
    objControladorCRUDEst.editarRegistro(nc: numControl, nuevoNom: nuevoNombre, nuevoApe: nuevoApellido, nuevoSem: nuevoSemestre, nuevoGpo: nuevoGrupo, nuev.  
  
    limpiarCampos();  
}  
  
private void rdbtnGrupoAKeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
    if (evt.getKeyCode() == KeyEvent.VK_ENTER)  
        this.cboCarrera.requestFocus();  
}
```

# FRM MODIFICAR ESTUDIANTE

```
private void txtNombreKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.txtApellidos.requestFocus();
}

private void cboCarreraKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.btnActualizar.requestFocus();
}

private void txtApellidosKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.cboSemestre.requestFocus();
}

private void rdbtnGrupoBKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.cboCarrera.requestFocus();
}

private void btnBuscarMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    this.buscar();
}

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

```
private void btnBuscarKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.buscar();
}

private void rdbtnGrupoCKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.cboCarrera.requestFocus();
}

private void txtNumControlKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyCode()==KeyEvent.VK_ENTER)
        this.txtNombre.requestFocus();
}

private void limpiarSeleccionGrupo() {
    // Limpiar todos los botones de grupo (asegurándonos que ninguno esté seleccionado)
    rdbtnGrupoA.setSelected(b: false);
    rdbtnGrupoB.setSelected(b: false);
    rdbtnGrupoC.setSelected(b: false);
}
```



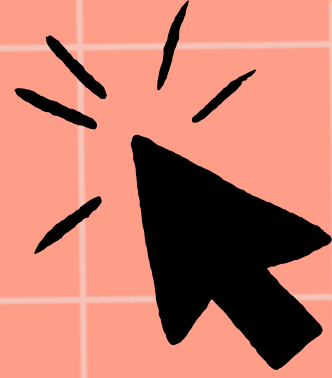
# FRM MODIFICAR ESTUDIANTE

```
private void limpiarSeleccionGrupo() {  
    // Limpiar todos los botones de grupo (asegurándonos que ninguno esté seleccionado)  
    rdbtnGrupoA.setSelected(b: false);  
    rdbtnGrupoB.setSelected(b: false);  
    rdbtnGrupoC.setSelected(b: false);  
}  
  
private void buscar(){  
    String numControl = this.txtNumControl.getText();  
    String registro[] = this.objControladorCRUDEst.buscarRegistro(nc: numControl);  
    if (registro != null) {  
        this.txtNombre.setText(registro[1]);  
        this.txtApellidos.setText(registro[2]);  
        this.cboSemestre.setSelectedItem(anObject: String.valueOf(i: Integer.parseInt(registro[3])));  
        this.rdbtnGrupoA.setSelected(b: false);  
        this.rdbtnGrupoB.setSelected(b: false);  
        this.rdbtnGrupoC.setSelected(b: false);  
  
        switch (registro[4]) {  
            case "A":  
                this.rdbtnGrupoA.setSelected(b: true);  
                break;  
            case "B":  
                this.rdbtnGrupoB.setSelected(b: true);  
                break;  
            case "C":  
                this.rdbtnGrupoC.setSelected(b: true);  
                break;  
        }  
        this.cboCarrera.setSelectedItem(registro[5]);  
        this.btnActualizar.setEnabled(b: true);  
    } else {  
        JOptionPane.showMessageDialog(parentComponent: this, message: "El registro no Existe");  
    }  
}
```

# FRM MODIFICAR ESTUDIANTE

```
// Variables declaration - do not modify
private javax.swing.JButton btnActualizar;
private javax.swing.JButton btnBuscar;
private javax.swing.JComboBox<String> cboCarrera;
private javax.swing.JComboBox<String> cboSemestre;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JRadioButton rdbtnGrupoA;
private javax.swing.JRadioButton rdbtnGrupoB;
private javax.swing.JRadioButton rdbtnGrupoC;
private javax.swing.JTextField txtApellidos;
private javax.swing.JTextField txtNombre;
private javax.swing.JTextField txtNumControl;
// End of variables declaration
}
```

# PROYECTO EN EJECUCIÓN(AGREGAR)



Agenda de Estudiantes

**Tecnológico del Valle de Oaxaca**  
**Agenda de Estudiantes**



Num.Control	Nombre	Apellidos	Semestre	Grupo	Carrera
23920001	Teresa	Santiago Jar...	12	B	Ing. en Cienc...
23920002	lirio	santi	01	A	Ing. Informáti...
23920003	esperanza	santiago	03	A	Ing. Informáti...
23920005	Flor	rodri	05	B	Ing. en Gesti...
23920200	mar	arena	01	A	Ing. Informáti...
23920202	lirio	lopez	01	A	Ing. Informáti...
23920205	karen	ruiz	01	B	Ing. Informáti...
23920211	karen	ruiz	01	A	Ing. Informáti...
23920217	Mareli	Aragón Anton...	04	A	Ing. Informáti...

Agenda de Estudiantes

**Tecnológico del Valle de Oaxaca**  
**Registrar Estudiantes**

Num Control


Nombre

Apellidos

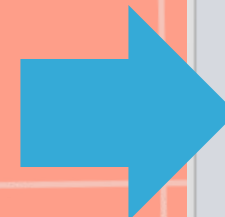
Semestre

Grupo ☒ A ☐ B ☐ C

Carrera

 **Guardar**

Num.Control	Nombre	Apellidos	Semestre	Grupo	Carrera
23920003	esperanza	santiago	03	A	Ing. Informáti...
23920005	Flor	rodri	05	B	Ing. en Gesti...
23920200	mar	arena	01	A	Ing. Informáti...
23920202	lirio	lopez	01	A	Ing. Informáti...
23920205	karen	ruiz	01	B	Ing. Informáti...
23920211	karen	ruiz	01	A	Ing. Informáti...
23920217	Mareli	Aragón Anton...	04	A	Ing. Informáti...
23920233	Radai	Manzano Ra...	04	A	Ing. Informáti...
23920248	karen	ruiz	04	A	Ing. Informáti...





# PROYECTO EN EJECUCIÓN(ELIMINAR)



Eliminar Registro

**Tecnológico del Valle de Oaxaca**

**Eliminar Estudiante**

Num Control

Nombre

Apellidos

Semestre

Grupo ☒ A ☐ B ☐ C

Carrera



 **Eliminar**

Agenda de Estudiantes

**Tecnológico del Valle de Oaxaca**

**Agenda de Estudiantes**

Num.Control	Nombre	Apellidos	Semestre	Grupo	Carrera
23920001	Teresa	Santiago Jar...	12	B	Ing. en C
23920002	lirio	santi	01	A	Ing. Infor
23920003	esperanza	santiago	03	A	Ing. Infor
23920005	Flor	rodri	05	B	Ing. en G
23920200	mar	arena	01	A	Ing. Infor
23920202	lirio	lopez	01	A	Ing. Infor
23920205	karen	ruiz	01	B	Ing. Infor
23920211	karen	ruiz	01	A	Ing. Infor
23920217	M...	A...	01	A	Ing. Infor

Eliminar Registro

**Tecnológico del Valle de Oaxaca**

**Eliminar Estudiante**

Num Control


Nombre

Apellidos


Semestre

Grupo ☒ A ☐ B ☐ C

Carrera

 **Eliminar**

Select an Option

 Estas seguro de eliminar el registro

# PROYECTO EN EJECUCIÓN(ELIMINAR)



Agenda de Estudiantes

Tecnológico del Valle de Oaxaca

Agenda de Estudiantes

Num.Control	Nombre	Carrera
23920001	Teresa	Ing. en Cie
23920002	lirio	Ing. Inform
23920003	esperan	Ing. Inform
23920005	Flor	Ing. en Ge
23920200	mar	Ing. Inform
23920205	karen	Ing. Inform
23920211	karen	Ing. Inform
23920217	Mareli	Ing. Inform

Message

 Registro Eliminado

OK

Eliminar Registro

Tecnológico del Valle de Oaxaca

Eliminar Estudiante

Num Control



Nombre

Apellidos

Semestre


1

Grupo

☒ A ☐ B ☐ C

Carrera

Ing. Informática




 Eliminar

Agenda de Estudiantes

Tecnológico del Valle de Oaxaca

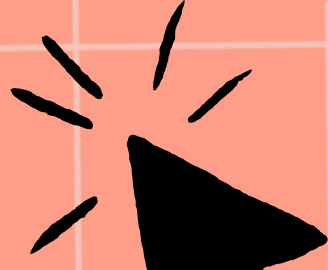
Agenda de Estudiantes

Num.Control	Nombre	Apellidos	Semestre	Grupo	Carrera
23920001	Teresa	Santiago Jar...	12	B	Ing. en Cienc...
23920002	lirio	santi	01	A	Ing. Informáti...
23920003	esperanza	santiago	03	A	Ing. Informáti...
23920005	Flor	rodri	05	B	Ing. en Gesti...
23920200	mar	arena	01	A	Ing. Informáti...
23920205	karen	ruiz	01	B	Ing. Informáti...
23920211	karen	ruiz	01	A	Ing. Informáti...
23920217	Mareli	Aragón Anton...	04	A	Ing. Informáti...



# PROYECTO EN EJECUCIÓN(MODIFICAR)



Tecnológico del Valle de Oaxaca

Modificar Estudiante

Num Control

23920005

Nombre

Flor

Apellidos

rodri

Semestre

5

▼

Grupo

☐ A ☒ B ☐ C

Carrera

Ing. en Gestión Empr...

▼

Actualizar

Agenda de Estudiantes

Num.Control

Nombre

Apellidos

Semestre

Grupo

Carrera

23920001	Teresa	Santiago Jar...	12	B	Ing. en Cienc...
23920002	lirio	santi	01	A	Ing. Informáti...
23920003	esperanza	santiago	03	A	Ing. Informáti...
23920005	Flor	rodri	05	B	Ing. en Gesti...
23920200	mar	arena	01	A	Ing. Informáti...
23920205	karen	ruiz	01	B	Ing. Informáti...
23920211	karen	ruiz	01	A	Ing. Informáti...
23920217	Mareli	Aragón Anton...	04	A	Ing. Informáti...
23920222	Patricia	Martínez B...	04	A	Ing. Informáti...

Tecnológico del Valle de Oaxaca

Modificar Estudiante

Num Control

23920005

Nombre

Flor

Apellidos

santiago

Semestre

5

▼

Grupo

☐ A ☒ B ☐ C

Carrera

Ing. en Gestión Empr...

▼

Actualizar

Num.Control	Nombre	Apellidos	Semestre	Grupo	Carrera
23920001	Teresa	Santiago Jar...	12	B	Ing. en Cienc...
23920002	lirio	santi	01	A	Ing. Informáti...
23920003	esperanza	santiago	03	A	Ing. Informáti...
23920005	Flor	santiago	05	B	Ing. en Gesti...
23920200	mar	arena	01	A	Ing. Informáti...
23920205	karen	ruiz	01	B	Ing. Informáti...
23920211	karen	ruiz	01	A	Ing. Informáti...
23920217	Mareli	Aragón Anton...	04	A	Ing. Informáti...
23920222	Patricia	Martínez B...	04	A	Ing. Informáti...

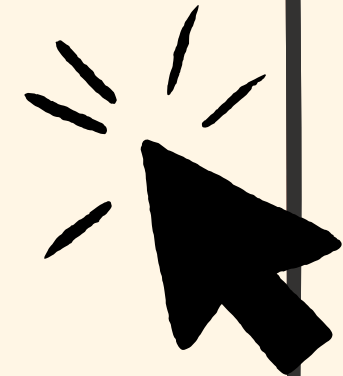


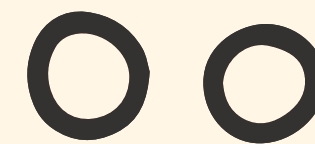


# LINK DONDE SE ALOJA DICHO PROYECTO.

[https://github.com/maki549/web-proyectosjava/blob/main/CRUDEstudiantesRAF\\_11.zip](https://github.com/maki549/web-proyectosjava/blob/main/CRUDEstudiantesRAF_11.zip)

**O CLIC AQUÍ**





# CONCLUSIÓN

En conclusión, este sistema CRUD de estudiantes representa una solución funcional y estructurada para la gestión académica básica. Gracias a la implementación de archivos con acceso directo (RandomAccessFile), se logró un manejo eficiente de la información sin necesidad de bases de datos externas.

Las clases han sido organizadas adecuadamente según su rol dentro del modelo vista-controlador (MVC), lo que facilita el mantenimiento del código y la separación de responsabilidades. La interfaz gráfica proporciona una experiencia de usuario clara y sencilla, mientras que la lógica del controlador y el acceso a archivos en el modelo aseguran la integridad y persistencia de los datos.

Este proyecto no solo cumple con los objetivos planteados, sino que también sienta las bases para un sistema más complejo que podría integrarse con bases de datos relacionales, autenticación de usuarios, o incluso conectividad en red.

