**UNIVERSIDAD MAYOR DE SAN ANDRÉS**
**FACULTAD DE CIENCIAS PURAS Y NATURALES**
**CARRERA INFORMATICA**


# PROGRAMACION II

## GRUPO 22: PROYECTO GESTIÓN DE UNA BIBLIOTECA

**Integrantes:** ALBERTH SAUL MAMANI PITA
YOVANI VICTOR CONTRERAS CALVIMONTES
GABRIEL ENRIQUE QUIROZ VILLCA
ALAN ALVIN PONCE MONTERO
JOSUE ELIAS MORALES ESPINOZA
JAVIER AVERANGA QUELKA

**Docente:** ROSALIA LOPEZ M.

**GitHub:**

**LA PAZ - BOLIVIA**

## INTRODUCCION

En este informe se desarrollará todo lo aprendido durante el curso de temporada del 2025 de la materia de Programación II.

Se implementará todos los conceptos de la Programación Orientada a Objetos, comúnmente llamada “POO”. Estos a su vez son: la creación de clases, la abstracción, los diferentes tipos de asociaciones tales como la herencia, agregación y composición, además de la persistencia, manejo de excepciones y la base de datos utilizando MySQL. Todo esto acompañado de una interfaz gráfica de la librería “Java Swing”.

## 1. Definición del Proyecto

### 1.1 Descripción General

El proyecto se basa en la gestión de una Biblioteca. Llevándola a código con el objetivo de simular cómo se haría dicha gestión en la vida real.
En la gestión de una biblioteca se tiene las siguientes funciones:

La administración de libros y sus derivados, estas funciones son:

- Agregar un libro a la biblioteca.
- Modificar un libro de la biblioteca.
- Eliminar un libro de la biblioteca.

También tienen funcionalidades los libros, estos son:

- Buscar los libros según su tipo, su autor, su título, su código ISBN, etc.

Por otro lado, también verificamos el estado de los libros, si se encuentran prestados o disponibles.

### 1.2 Objetivos

- Desarrollar un sistema de gestión de biblioteca utilizando Java y MySQL que facilite la administración de préstamos, devoluciones, y búsqueda de libros.

- Implementar una interfaz gráfica de usuario en Java que sea intuitiva y fácil de usar para los administradores y usuarios de la biblioteca.

- Automatizar las operaciones de préstamos y devoluciones, asegurando el estados de los libros.

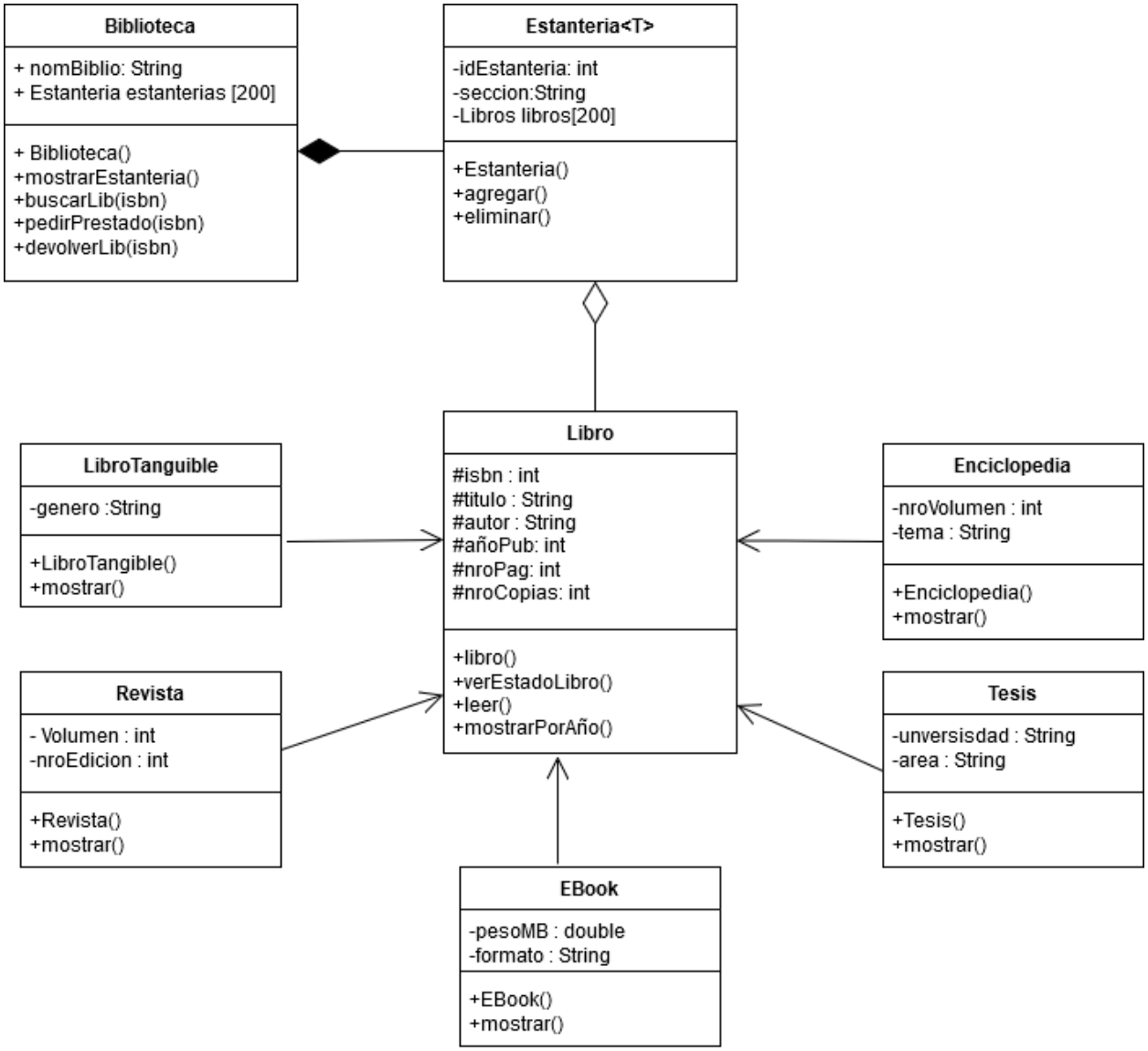
- Crear funcionalidades para consultar y gestionar los libros en las estanterías.

- Implementar medidas básicas de seguridad para proteger los datos almacenados en la base de datos.

## 2. ANALISIS Y DISENO

### 2.1 DIAGRAMA DE UML

El siguiente diagrama presentará todo lo necesario de una clase UML.




### 2.2 PRINCIPIOS DE DISEÑO

Clases: Creación de las plantillas principales Libro, Biblioteca y Estanteria.
Herencia: Las clases Revista, EBook, Enciclopedia, LibroTangible y Tesis son herencia de la clase Libro.
Composición: La clase Biblioteca se compone de Estantería, ya que necesita de estos para poder permanecer.

Agregación: Un libro puede seguir existiendo incluso si la Biblioteca se quiebra por ejemplo.
Genericidad: Uso de colecciones genéricas para gestionar listas de Estanterías donde almacenará los libros.
Interfaces: Implementación de una interfaz Gráfica (GUI) con el uso de la librería de Java Swing.

Persistencia: Donde nuestros libros tienen que persistir con el tiempo, ya sea si se agregan, borran o modifican. Esto se hará con una base de datos MySQL.

## 3. Implementación en Java

### 3.1 Estructura del proyecto

El proyecto está organizado en un único paquete llamado Biblioteca, que contiene las clases necesarias para modelar y gestionar los materiales bibliográficos en una biblioteca. El diseño sigue principios de la programación orientada a objetos, como la herencia y el polimorfismo, lo que permite una estructura modular y extensible.

**Paquete Biblioteca:**

1. **Libro (Clase Base):**
   La clase Libro es la base para todos los tipos de materiales bibliográficos en la biblioteca. Define los atributos y métodos comunes a todos los libros, como el título, autor, ISBN, año de publicación, número de páginas, número de copias y edad recomendada. Además, incluye métodos para reducir y aumentar las copias disponibles del libro, y para mostrar o leer la información del libro.
   **Atributos:**
   - titulo: Título del libro.
   - autor: Autor del libro.
   - ISBN: Código único de identificación.
   - anioPublicacion: Año de publicación del libro.
   - nroPaginas: Número de páginas del libro.
   - nroCopias: Número de copias disponibles.
   - edadClasificacion: Edad recomendada para el libro.
2. **Métodos:**
   - mostrar(): Muestra los atributos del libro.
   - leer(): Solicita los datos del libro mediante cuadros de diálogo.
   - reducirCopias(): Reduce el número de copias disponibles.
   - aumentarCopias(): Aumenta el número de copias disponibles.
3. **Clases Derivadas de Libro:** Estas clases heredan de Libro y agregan atributos específicos para cada tipo de material bibliográfico:
   - **Tesis:** Representa una tesis académica. Atributos adicionales:

        - universidad: Universidad de origen de la tesis.
        - area: Área de estudio de la tesis.
    - **Revista:** Representa una revista. Atributos adicionales:
        - volumen: Volumen de la revista.
        - nroEdicion: Número de edición de la revista.
    - **LibroTangible:** Representa un libro físico. Atributo adicional:
        - genero: Género literario del libro.
    - **Enciclopedia:** Representa una enciclopedia. Atributos adicionales:
        - nroEdicion: Número de edición de la enciclopedia.
        - tema: Tema o área de conocimiento de la enciclopedia.
    - **EBook:** Representa un libro electrónico. Atributos adicionales:
        - pesoMB: Peso del archivo en megabytes.
        - formato: Formato del archivo (PDF, EPUB, etc.).
4. Cada una de estas clases sobrescribe los métodos mostrar() y leer() de la clase base Libro para adaptarlos a sus necesidades específicas.
5. **Estanteria<T extends Libro>:** La clase Estanteria es genérica y permite gestionar una estantería que puede contener cualquier tipo de Libro. Los métodos de esta clase permiten agregar libros, mostrar los libros en la estantería, realizar devoluciones, verificar si un libro está disponible para préstamo, eliminar libros, contar las copias disponibles y buscar libros por autor.
   **Atributos:**
    - ID: Identificador único de la estantería.
    - sección: Nombre de la sección de la estantería.
    - nroLibros: Número de libros en la estantería.
    - libros: Lista de libros en la estantería.
6. **Métodos:**
    - addLibro(): Agrega un libro a la estantería.
    - mostrarEstanteria(): Muestra todos los libros en la estantería.
    - devolucionLibro(): Permite la devolución de un libro.
    - isDisponible(): Verifica si un libro está disponible para préstamo.
    - eliminarLibro(): Elimina un libro de la estantería.
    - nroCopiasLibro(): Obtiene el número de copias disponibles de un libro.
    - buscarAutor(): Busca libros por autor en la estantería.

### 3.2 Código Fuente

1. **Clase Base: Biblioteca**

```
public class Biblioteca {
    private Estanteria<Libro> estanteria;
    private JTextArea outputArea;

    public Biblioteca() {
```

```
        estanteria <- new Estanteria<>();
        outputArea <- new JTextArea(10, 30);
    }

    public void mostrarGUI() {
        JFrame frame <- new JFrame("Biblioteca");
        JButton btnAgregarLibro <- new JButton("Agregar Libro");
        JButton btnListarLibros <- new JButton("Listar Libros");
        JButton btnBuscarAutor <- new JButton("Buscar por Autor");
        JButton btnBuscarTitulo <- new JButton("Buscar por Título");
        JButton btnBuscarISBN <- new JButton("Buscar por ISBN");
        JButton btnBuscarNroCopias <- new JButton("Buscar por Nro. de Copias");
        JButton btnPrestar <- new JButton("Prestar Libro");
        JButton btnDevolver <- new JButton("Devolver Libro");

        btnAgregarLibro.addActionListener(e -> agregarLibro());
        btnListarLibros.addActionListener(e -> listarLibros());
        btnBuscarAutor.addActionListener(e -> buscarPorAutor());
        btnBuscarTitulo.addActionListener(e -> buscarPorTitulo());
        btnBuscarISBN.addActionListener(e -> buscarPorISBN());
        btnBuscarNroCopias.addActionListener(e -> buscarPorNroCopias());
        btnPrestar.addActionListener(e -> prestarLibro());
        btnDevolver.addActionListener(e -> devolverLibro());

        frame.setLayout(new BoxLayout(frame.getContentPane(),
BoxLayout.Y_AXIS));
        frame.add(new JScrollPane(outputArea));
        frame.add(btnAgregarLibro);
        frame.add(btnListarLibros);
        frame.add(btnBuscarAutor);
        frame.add(btnBuscarTitulo);
        frame.add(btnBuscarISBN);
        frame.add(btnBuscarNroCopias);
        frame.add(btnPrestar);
        frame.add(btnDevolver);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }

    private void agregarLibro() {
        String[] opciones <- {"Libro Tangible", "EBook", "Revista", "Enciclopedia",
"Tesis"};
        int eleccion <- JOptionPane.showOptionDialog(null, "Seleccione el tipo de
libro:", "Agregar Libro",
```

```
            JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE, null, opciones, opciones[0]);

        if (eleccion == -1) return;

        try {
            int isbn <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el
ISBN del libro:"));
            String titulo <- JOptionPane.showInputDialog("Ingrese el título del libro:");
            String autor <- JOptionPane.showInputDialog("Ingrese el autor del
libro:");
            int anioPub <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el
año de publicación:"));
            int nroPag <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el
número de páginas:"));
            int nroCopias <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese
el número de copias disponibles:"));

            Libro libro <- null;
            switch (eleccion) {
                case 0 -> {
                    String genero <- JOptionPane.showInputDialog("Ingrese el género
del libro tangible:");
                    libro <- new LibroTangible(isbn, titulo, autor, anioPub, nroPag,
nroCopias, genero);
                }
                case 1 -> {
                    double pesoMB <-
Double.parseDouble(JOptionPane.showInputDialog("Ingrese el peso en MB del
eBook:"));
                    String formato <- JOptionPane.showInputDialog("Ingrese el formato
del eBook (e.g., PDF, EPUB):");
                    libro <- new EBook(isbn, titulo, autor, anioPub, nroPag, nroCopias,
pesoMB, formato);
                }
                case 2 -> {
                    int volumen <-
Integer.parseInt(JOptionPane.showInputDialog("Ingrese el volumen de la
revista:"));
                    int nroEdicion <-
Integer.parseInt(JOptionPane.showInputDialog("Ingrese el número de edición de
la revista:"));
                    libro <- new Revista(isbn, titulo, autor, anioPub, nroPag, nroCopias,
volumen, nroEdicion);
                }
                case 3 -> {
```

```
                int nroVolumen <-
Integer.parseInt(JOptionPane.showInputDialog("Ingrese el número de volumen
de la enciclopedia:"));
                String tema <- JOptionPane.showInputDialog("Ingrese el tema de la
enciclopedia:");
                libro <- new Enciclopedia(isbn, titulo, autor, anioPub, nroPag,
nroCopias, nroVolumen, tema);
            }
            case 4 -> {
                String universidad <- JOptionPane.showInputDialog("Ingrese la
universidad asociada a la tesis:");
                String area <- JOptionPane.showInputDialog("Ingrese el área de
investigación de la tesis:");
                libro <- new Tesis(isbn, titulo, autor, anioPub, nroPag, nroCopias,
universidad, area);
            }
        }

        if (libro != null) {
            estanteria.agregarLibro(libro);
            outputArea.append("Libro agregado: " + libro + "\n");
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(null, "Por favor ingrese valores
numéricos válidos.");
    }
}

private void listarLibros() {
    outputArea.setText(estanteria.listarLibros());
}

private void buscarPorAutor() {
    String autor <- JOptionPane.showInputDialog("Ingrese el autor a buscar:");
    List<Libro> resultados <- estanteria.buscarPorAutor(autor);
    mostrarResultadosBusqueda(resultados);
}

private void buscarPorTitulo() {
    String titulo <- JOptionPane.showInputDialog("Ingrese el título a buscar:");
    List<Libro> resultados <- estanteria.buscarPorTitulo(titulo);
    mostrarResultadosBusqueda(resultados);
}

private void buscarPorISBN() {
```

```
        int isbn <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el ISBN
a buscar:"));
        Libro resultado <- estanteria.buscarPorISBN(isbn);
        outputArea.setText(resultado != null ? resultado.toString() : "No se encontró
el libro con ese ISBN.");
    }

    private void buscarPorNroCopias() {
        int nroCopias <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el
número de copias a buscar:"));
        List<Libro> resultados <- estanteria.buscarPorNroCopias(nroCopias);
        mostrarResultadosBusqueda(resultados);
    }

    private void prestarLibro() {
        int isbn <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el ISBN
del libro a prestar:"));
        boolean exito <- estanteria.prestar(isbn);
        outputArea.setText(exito ? "Libro prestado exitosamente." : "No se pudo
prestar el libro.");
    }

    private void devolverLibro() {
        int isbn <- Integer.parseInt(JOptionPane.showInputDialog("Ingrese el ISBN
del libro a devolver:"));
        boolean exito <- estanteria.devolver(isbn);
        outputArea.setText(exito ? "Libro devuelto exitosamente." : "No se pudo
devolver el libro.");
    }

    private void mostrarResultadosBusqueda(List<Libro> resultados) {
        if (resultados.isEmpty()) {
            outputArea.setText("No se encontraron resultados.");
        } else {
            StringBuilder sb <- new StringBuilder();
            for (Libro libro : resultados) {
                sb.append(libro.toString()).append("\n");
            }
            outputArea.setText(sb.toString());
        }
    }
}
```

2. **Clase Generica: Estanteria**

```
public class Estanteria<T extends Libro> {
    private List<T> libros <- new ArrayList<>();

    public void agregarLibro(T libro) {
        libros.add(libro);
    }

    public List<T> getLibros() {
        return libros;
    }

    public String listarLibros() {
        if (libros.isEmpty()) {
            return "No hay libros en la estantería.";
        }
        StringBuilder sb <- new StringBuilder();
        for (T libro : libros) {
            sb.append(libro.toString()).append("\n");
        }
        return sb.toString();
    }

    public List<T> buscarPorAutor(String autor) {
        List<T> resultados <- new ArrayList<>();
        for (T libro : libros) {
            if (libro.getAutor().equalsIgnoreCase(autor)) {
                resultados.add(libro);
            }
        }
        return resultados;
    }

    public List<T> buscarPorTitulo(String titulo) {
        List<T> resultados <- new ArrayList<>();
        for (T libro : libros) {
            if (libro.getTitulo().equalsIgnoreCase(titulo)) {
                resultados.add(libro);
            }
        }
        return resultados;
    }

    public T buscarPorISBN(int isbn) {
        for (T libro : libros) {
            if (libro.getIsbn() == isbn) {
```

```
                return libro;
            }
        }
        return null;
    }

    public List<T> buscarPorNroCopias(int nroCopias) {
        List<T> resultados <- new ArrayList<>();
        for (T libro : libros) {
            if (libro.getNroCopias() == nroCopias) {
                resultados.add(libro);
            }
        }
        return resultados;
    }

    public boolean prestar(int isbn) {
        T libro <- buscarPorISBN(isbn);
        if (libro != null && libro.getNroCopias() > 0) {
            libro.setNroCopias(libro.getNroCopias() - 1);
            return true;
        }
        return false;
    }

    public boolean devolver(int isbn) {
        T libro <- buscarPorISBN(isbn);
        if (libro != null) {
            libro.setNroCopias(libro.getNroCopias() + 1);
            return true;
        }
        return false;
    }
}
```

3. **Clase padre: Libro**

```
public class Libro {
    private int isbn;
    private String titulo;
    private String autor;
    private int anioPub;
    private int nroPag;
    private int nroCopias;

    public Libro(int isbn, String titulo, String autor, int anioPub, int nroPag, int
nroCopias) {
        this.isbn <- isbn;
```

```
        this.titulo <- titulo;
        this.autor <- autor;
        this.anioPub <- anioPub;
        this.nroPag <- nroPag;
        this.nroCopias <- nroCopias;
    }

    public int getIsbn() {
        return isbn;
    }

    public String getTitulo() {
        return titulo;
    }

    public String getAutor() {
        return autor;
    }

    public int getAnioPub() {
        return anioPub;
    }

    public int getNroPag() {
        return nroPag;
    }

    public int getNroCopias() {
        return nroCopias;
    }

    public void setNroCopias(int nroCopias) {
        this.nroCopias <- nroCopias;
    }

    @Override
    public String toString() {
        return "Título: " + titulo + ", Autor: " + autor + ", ISBN: " + isbn +
                ", Año: " + anioPub + ", Páginas: " + nroPag + ", Copias: " + nroCopias;
    }
}
```

4. **Clase hija: Libro Tangible**

```
public class LibroTangible extends Libro {
    private String genero;
```

```
    public LibroTangible(int isbn, String titulo, String autor, int anioPub, int nroPag,
int nroCopias, String genero) {
        super(isbn, titulo, autor, anioPub, nroPag, nroCopias);
        this.genero <- genero;
    }

    public String getGenero() {
        return genero;
    }

    @Override
    public String toString() {
        return super.toString() + ", Género: " + genero;
    }
}
```

5. **Clase hija: Revista**

```
public class Revista extends Libro {
    private int volumen;
    private int nroEdicion;

    public Revista(int isbn, String titulo, String autor, int anioPub, int nroPag, int
nroCopias, int volumen, int nroEdicion) {
        super(isbn, titulo, autor, anioPub, nroPag, nroCopias);
        this.volumen <- volumen;
        this.nroEdicion <- nroEdicion;
    }

    public int getVolumen() {
        return volumen;
    }

    public int getNroEdicion() {
        return nroEdicion;
    }

    @Override
    public String toString() {
        return super.toString() + ", Volumen: " + volumen + ", Edición: " +
nroEdicion;
    }
}
```

6. **Clase hija: EBook**

```
public class EBook extends Libro {
    private double pesoMB;
    private String formato;
```

```
    public EBook(int isbn, String titulo, String autor, int anioPub, int nroPag, int
nroCopias, double pesoMB, String formato) {
        super(isbn, titulo, autor, anioPub, nroPag, nroCopias);
        this.pesoMB <- pesoMB;
        this.formato <- formato;
    }

    public double getPesoMB() {
        return pesoMB;
    }

    public String getFormato() {
        return formato;
    }

    @Override
    public String toString() {
        return super.toString() + ", Peso: " + pesoMB + "MB, Formato: " + formato;
    }
}
```

7. **Clase hija: Enciclopedia**

```
public class Enciclopedia extends Libro {
    private int nroVolumen;
    private String tema;

    public Enciclopedia(int isbn, String titulo, String autor, int anioPub, int nroPag,
int nroCopias, int nroVolumen, String tema) {
        super(isbn, titulo, autor, anioPub, nroPag, nroCopias);
        this.nroVolumen <- nroVolumen;
        this.tema <- tema;
    }

    public int getNroVolumen() {
        return nroVolumen;
    }

    public String getTema() {
        return tema;
    }

    @Override
    public String toString() {
        return super.toString() + ", Volumen: " + nroVolumen + ", Tema: " + tema;
    }
}
```

8. **Clase hija: Tesis**

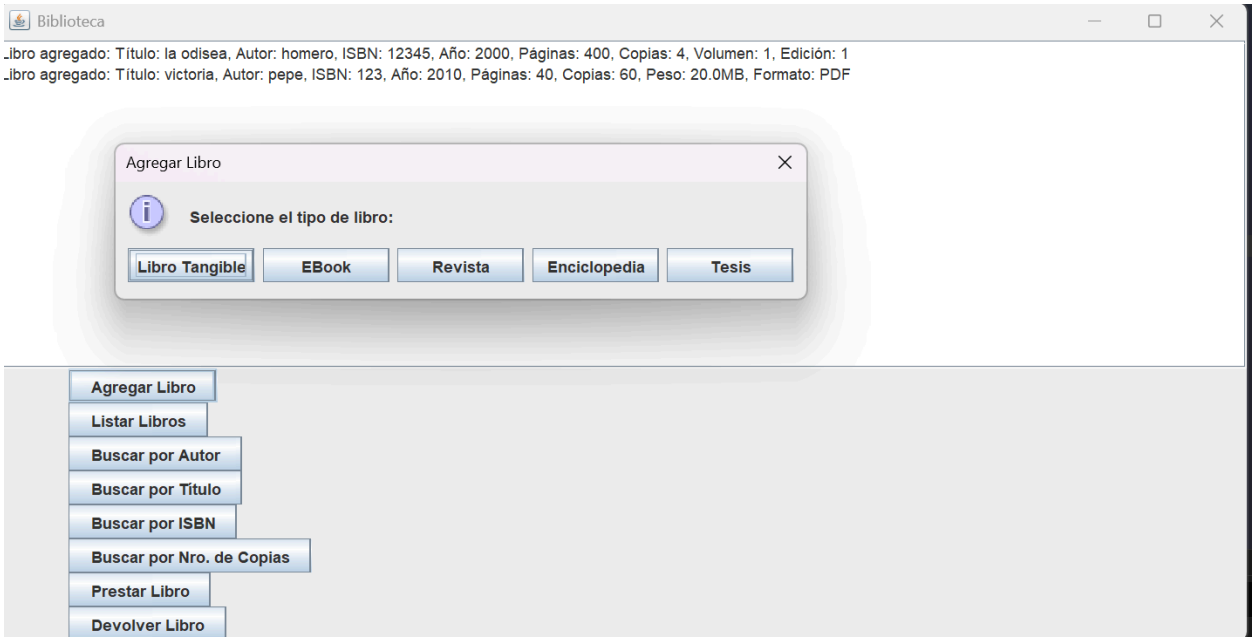
```
public class Tesis extends Libro {
    private String universidad;
    private String area;

    public Tesis(int isbn, String titulo, String autor, int anioPub, int nroPag, int
nroCopias, String universidad, String area) {
        super(isbn, titulo, autor, anioPub, nroPag, nroCopias);
        this.universidad <- universidad;
        this.area <- area;
    }

    public String getUniversidad() {
        return universidad;
    }

    public String getArea() {
        return area;
    }

    @Override
    public String toString() {
        return super.toString() + ", Universidad: " + universidad + ", Área: " + area;
    }
}
```

9. **Clase Principal**

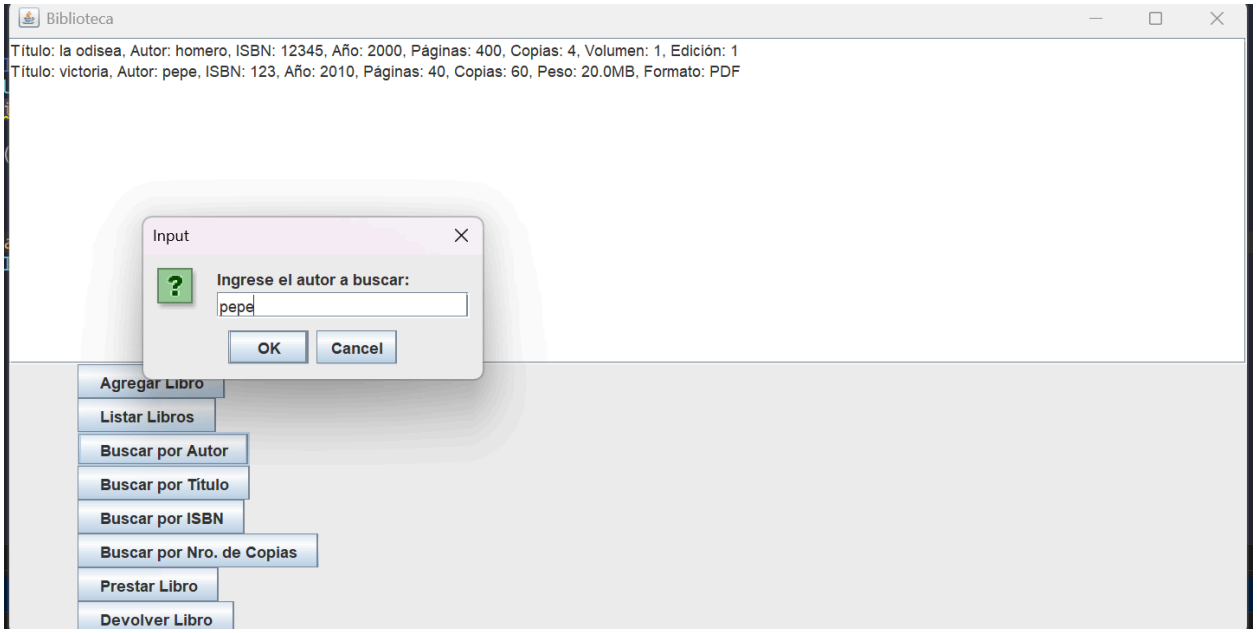
```
public class BibliotecaGUI {
    private Biblioteca biblioteca;

    public BibliotecaGUI() {
        biblioteca = new Biblioteca();
    }

    public static void main(String[] args) {
        new BibliotecaGUI().biblioteca.mostrarGUI();
    }
}
```

## 3.3 Diseño de interfaces

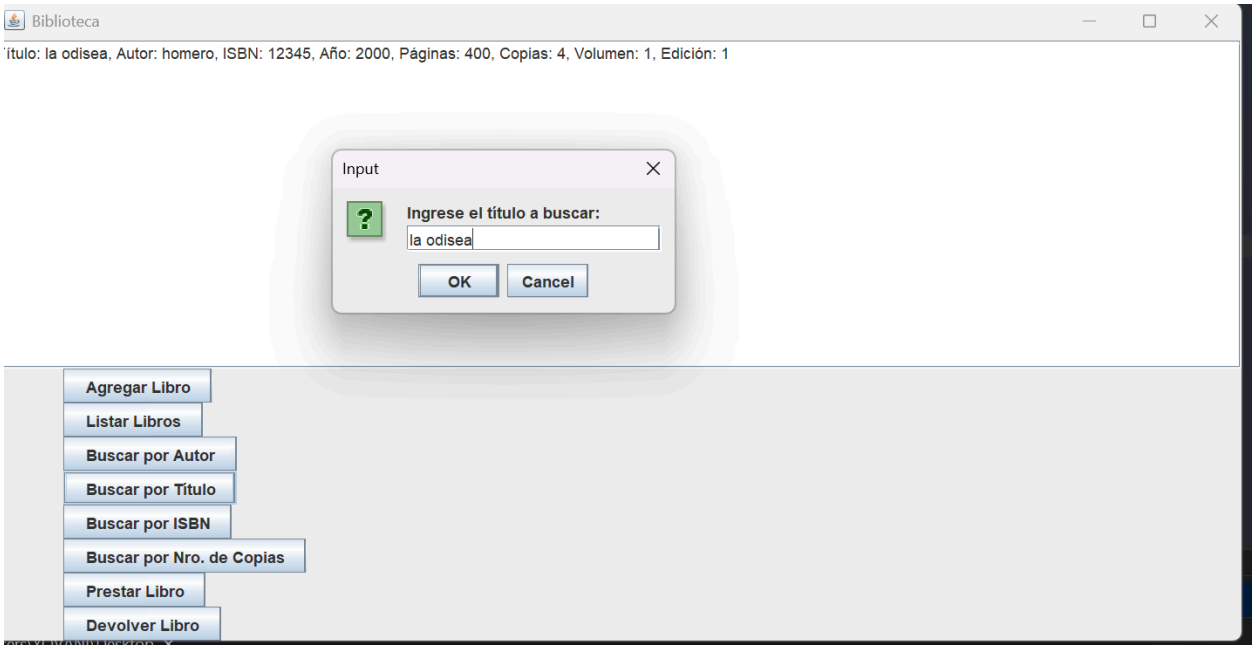
### 1. Agregar libro



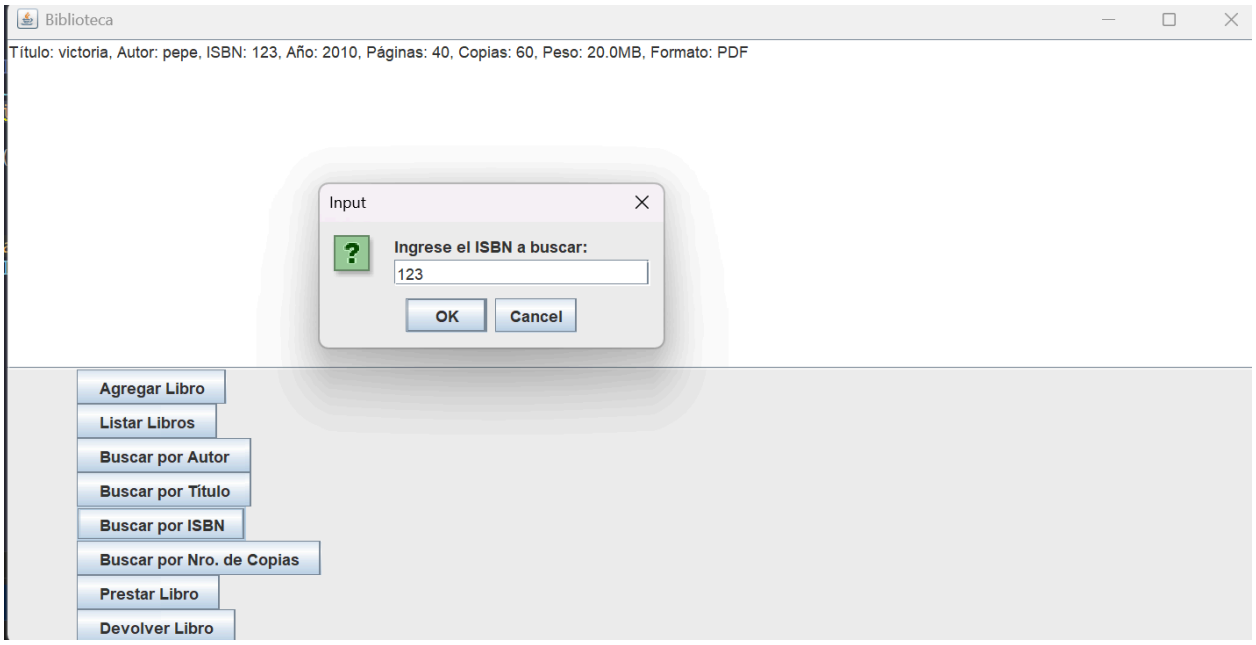

### 2. Buscar por autor



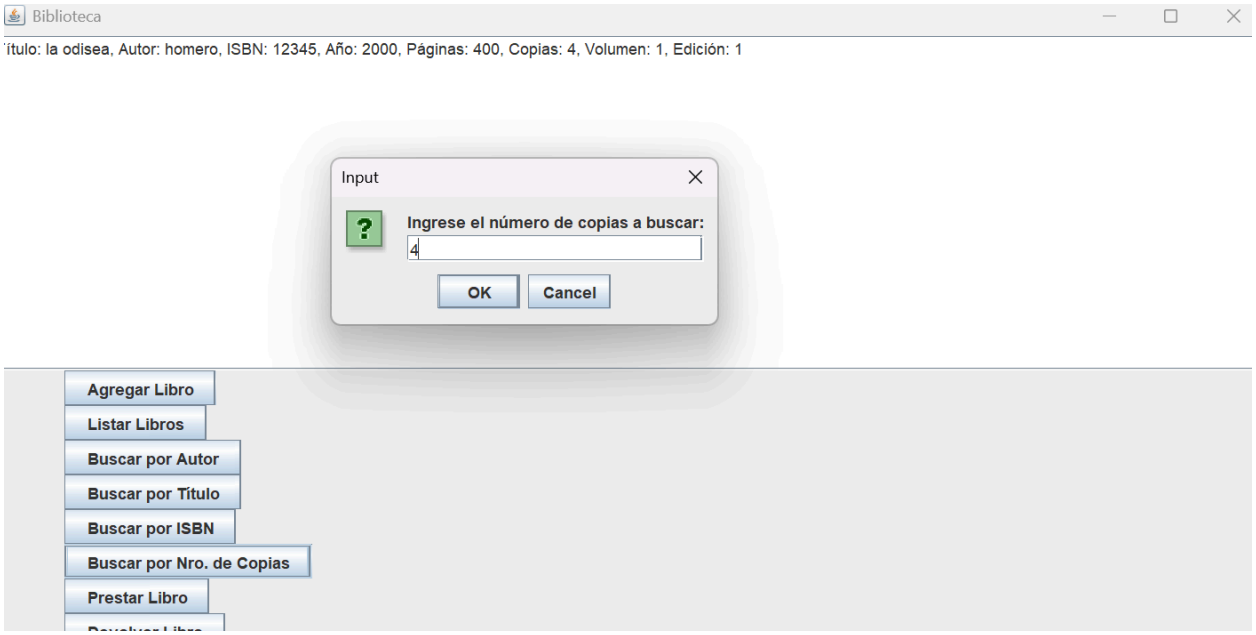
## 3. Buscar por título



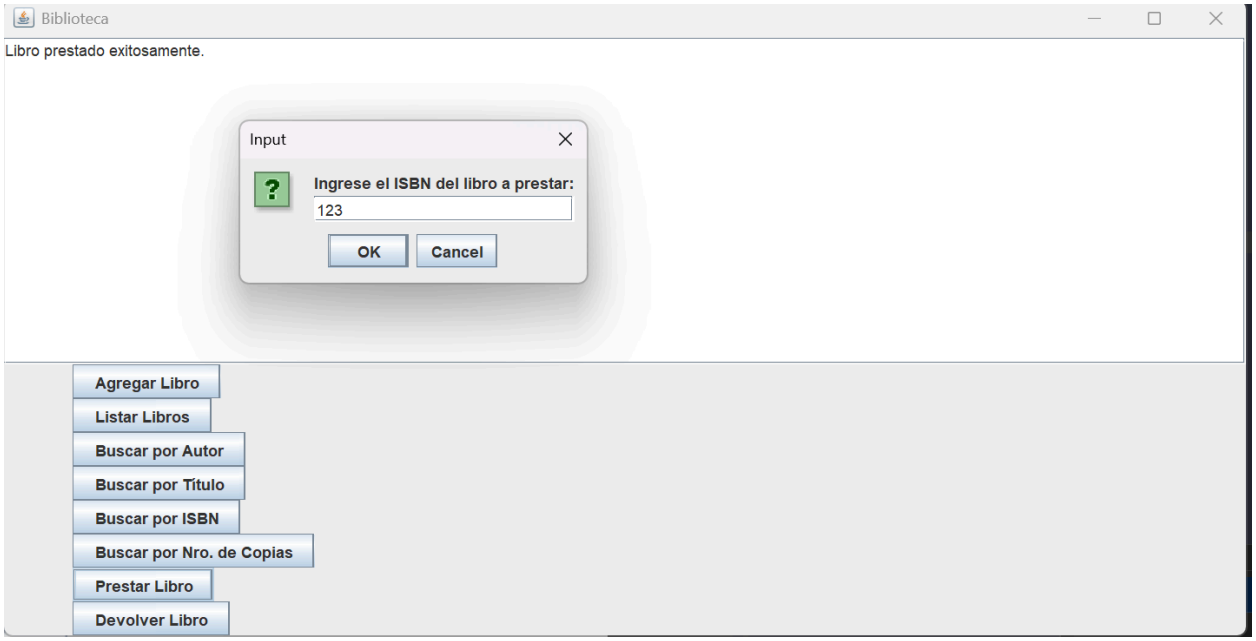

## 4. Buscar por isbn



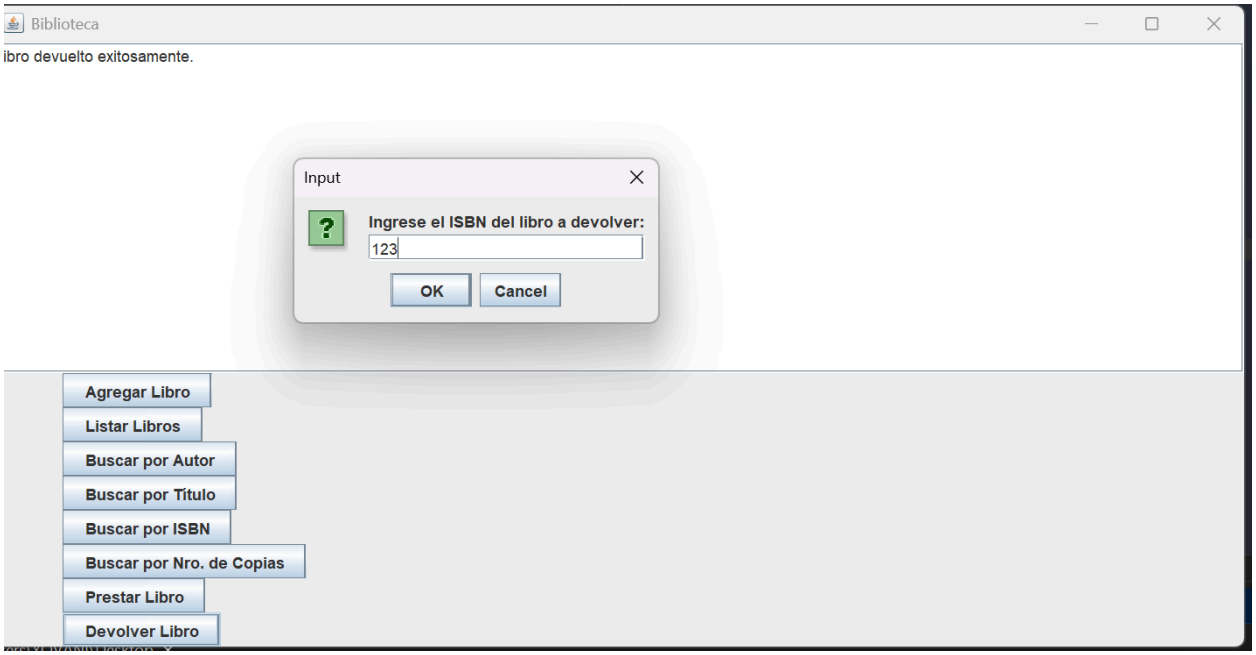
## 5. Buscar por copias




## 6. Prestar libro

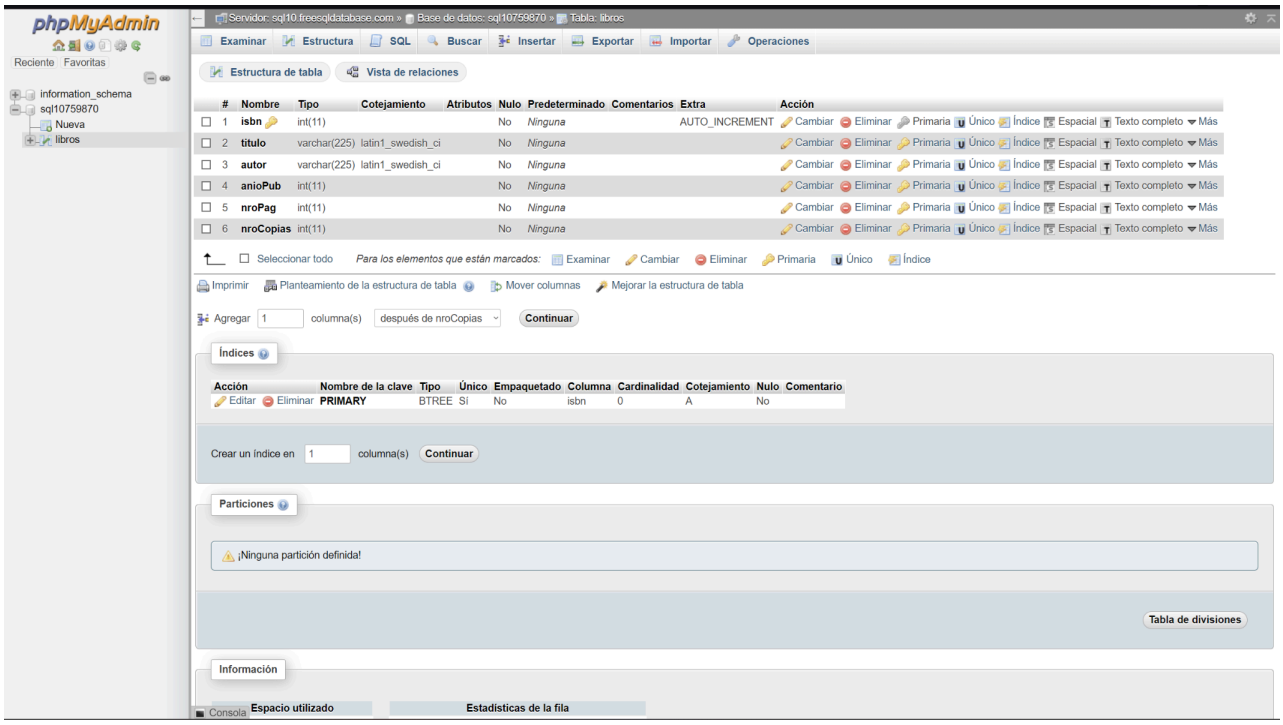


## 7. Devolver libro

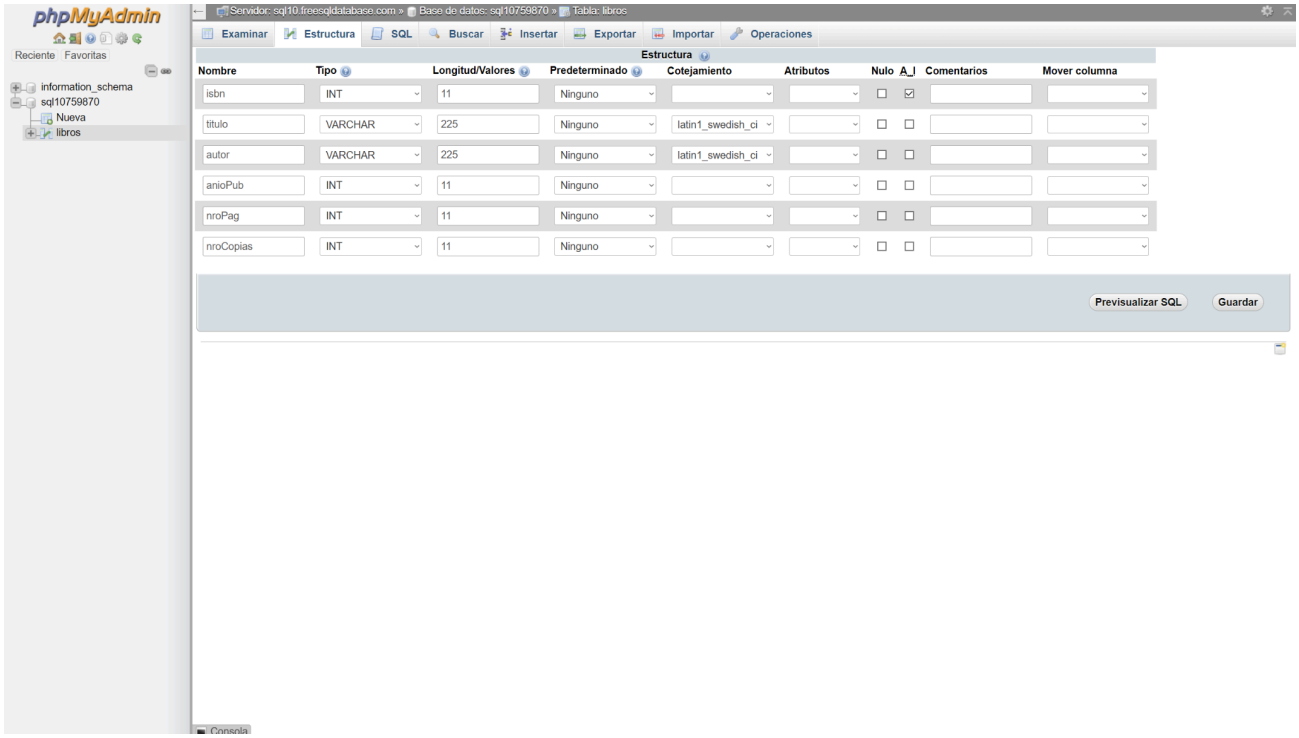



## 3.4 Manejo de archivos

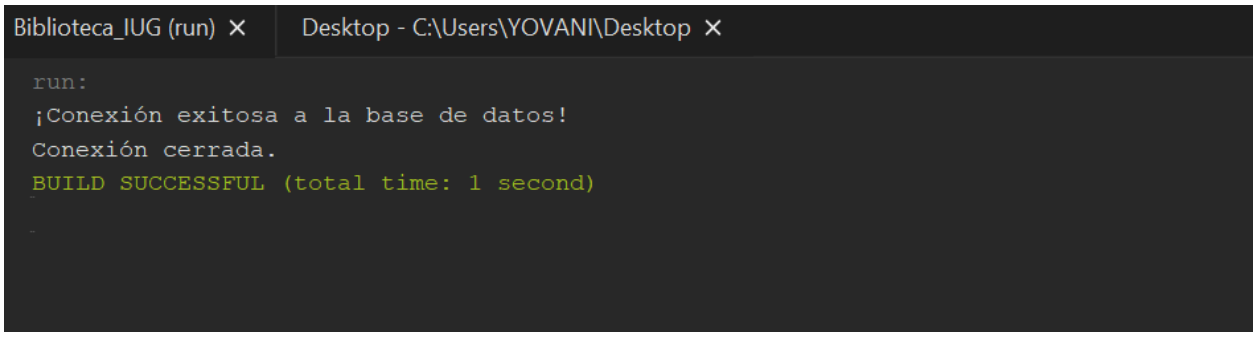
**1.**




**2.**




**3.**




## 4. Conclusion

El desarrollo de este proyecto de gestión de biblioteca permitió aplicar de manera integral los conceptos clave de la Programación Orientada a Objetos (POO), como herencia, genericidad y composición, junto con el uso de herramientas prácticas como MySQL y Java Swing para la persistencia y la interfaz gráfica, respectivamente. A través de este trabajo, se demostró la importancia de hacer una buena estructura y diseñar para lograr un sistema eficiente, accesible y fácil de usar. La experiencia adquirida durante este proyecto es valiosa para enfrentar futuros retos en el desarrollo de software con un enfoque en la solución de problemas reales y prácticos.