

HITO2\_PSP\_1T\_ALEJANDROPA  
WLUKIEWICZ

20/11/2024

## Contenido

|   |    |
|---|----|
| 1. Sistema de Persistencia de Datos.....            | 2  |
| Base de Datos .....                                 | 2  |
| ConexionBD .....                                    | 2  |
| Libreria .....                                      | 3  |
| 2. Capa de Acceso a Datos .....                     | 4  |
| LibroDAO .....                                      | 4  |
| 3. Arquitectura del Servidor .....                  | 5  |
| Componentes Principales.....                        | 5  |
| 4. Capacidad Multitarea .....                       | 9  |
| Implementación .....                                | 9  |
| Monitorización .....                                | 9  |
| 5. Gestión de Peticiones.....                       | 9  |
| Tipos de Peticiones Soportadas .....                | 10 |
| 6. Cliente .....                                    | 10 |
| Interfaz Gráfica.....                               | 10 |
| .....   | 16 |
| 7. Servidor .....                                   | 16 |
| Interfaz Gráfica.....                               | 16 |
| 8. Documentación Visual.....                        | 21 |
| Capturas de Pantalla .....                          | 21 |
| Pantalla de inicio. ....                            | 22 |
| INTERFAZ SERVIDOR.....                              | 22 |
| SERVIDOR INICIADO .....                             | 22 |
| SERVIDOR DETENIDO .....                             | 23 |
| INTERFAZ CLIENTE .....                              | 24 |
| Proceso de búsqueda. ....                           | 25 |
| Resultados filtrados.....                           | 26 |
| TODOS .....   | 26 |
| TITULO .....  | 26 |
| AUTOR .....   | 26 |
| PRECIO.....   | 27 |
| COMPROBACION DE UTILIZACION POR OTROS FILTROS ..... | 27 |
| Gestión de errores.....                             | 28 |
| SINO SE BUSCA NADA.....                             | 29 |
| SI SE BUSCA ALGO QUE NO ESTA EN LA BDD .....        | 29 |

|   |    |
|---|----|
| FUNCIONALIDAD .....   | 30 |
| SI SE LE DA AL BOTON LIMPIAR SE LIMPIA TANTO LOS RESULTADOS COMO EL CAMPO DE BUSQUEDA ..... | 30 |
| Webgrafía .....   | 30 |
| ANEXO I .....   | 30 |

# 1. Sistema de Persistencia de Datos

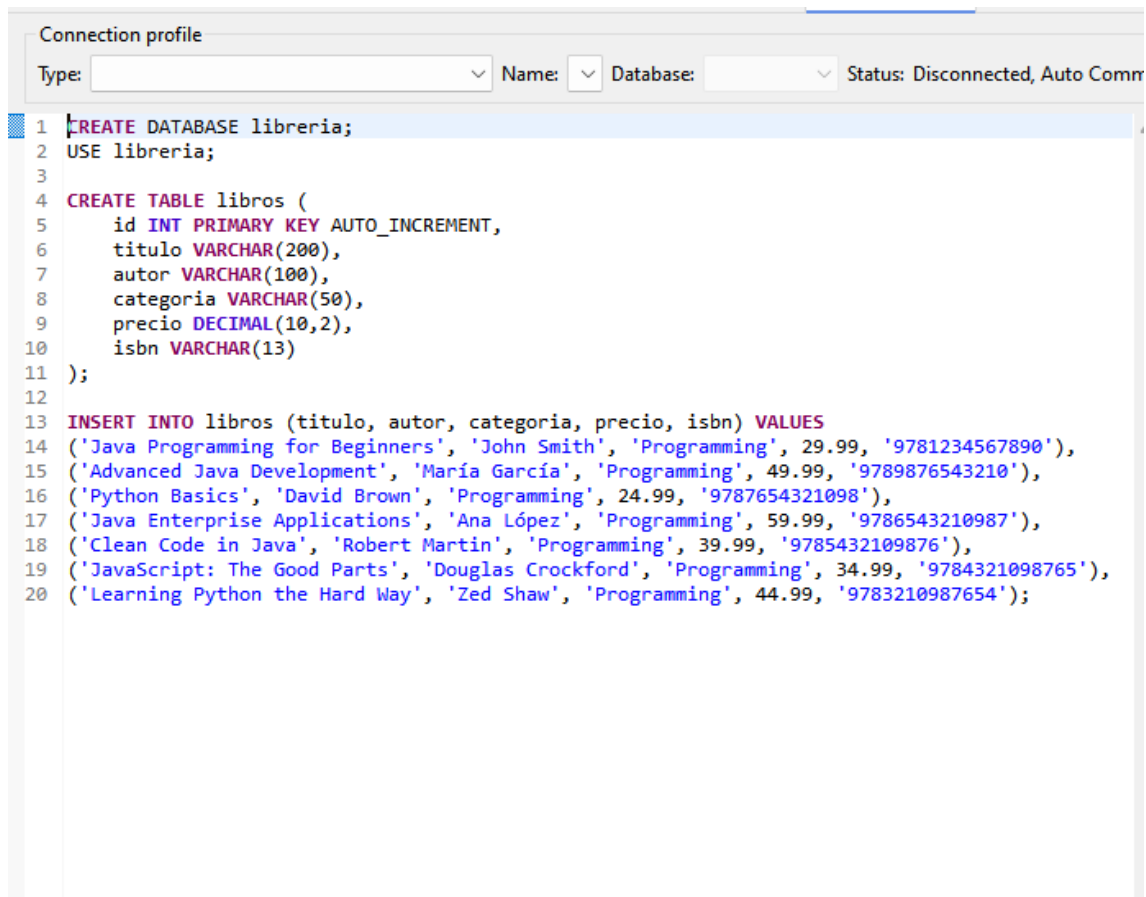
## Base de Datos

El sistema utiliza una base de datos MySQL llamada "libreria" para almacenar la información de los libros. La conexión a la base de datos se gestiona mediante la clase ConexionBD, que implementa una única instancia de conexión a la base de datos. Esta clase maneja las credenciales de acceso y proporciona métodos para obtener y cerrar conexiones de manera segura.

## ConexionBD

```
VehiculoTer... Vehiculo.java Coche.java Moto.java ConexionBD.java x »37
1 package datos;
2
3 import java.sql.Connection;
4
5
6
7 public class ConexionBD {
8     private static final String URL = "jdbc:mysql://localhost:3306/libreria";
9     private static final String USER = "root";
10    private static final String PASSWORD = "Tcachuk93";
11
12    static {
13        try {
14            // Registrar el driver explícitamente
15            Class.forName("com.mysql.cj.jdbc.Driver");
16            System.out.println("Driver MySQL registrado correctamente");
17        } catch (ClassNotFoundException e) {
18            System.err.println("Error al registrar el driver MySQL: " + e.getMessage());
19            throw new RuntimeException("No se pudo registrar el driver MySQL", e);
20        }
21    }
22
23    // Obtiene una conexión a la base de datos
24    public static Connection obtenerConexion() throws SQLException {
25        try {
26            Connection conexion = DriverManager.getConnection(URL, USER, PASSWORD);
27            System.out.println("Conexión establecida con éxito a la base de datos");
28            return conexion;
29        } catch (SQLException e) {
30            System.err.println("Error al conectar a la base de datos: " + e.getMessage());
31            throw e;
32        }
33    }
34 }
```

## Libreria



The screenshot shows a 'Connection profile' window with a text area containing SQL code. The code is as follows:

```
1 CREATE DATABASE libreria;
2 USE libreria;
3
4 CREATE TABLE libros (
5     id INT PRIMARY KEY AUTO_INCREMENT,
6     titulo VARCHAR(200),
7     autor VARCHAR(100),
8     categoria VARCHAR(50),
9     precio DECIMAL(10,2),
10    isbn VARCHAR(13)
11 );
12
13 INSERT INTO libros (titulo, autor, categoria, precio, isbn) VALUES
14 ('Java Programming for Beginners', 'John Smith', 'Programming', 29.99, '9781234567890'),
15 ('Advanced Java Development', 'María García', 'Programming', 49.99, '9789876543210'),
16 ('Python Basics', 'David Brown', 'Programming', 24.99, '9787654321098'),
17 ('Java Enterprise Applications', 'Ana López', 'Programming', 59.99, '9786543210987'),
18 ('Clean Code in Java', 'Robert Martin', 'Programming', 39.99, '9785432109876'),
19 ('JavaScript: The Good Parts', 'Douglas Crockford', 'Programming', 34.99, '9784321098765'),
20 ('Learning Python the Hard Way', 'Zed Shaw', 'Programming', 44.99, '9783210987654');
```

## 2. Capa de Acceso a Datos

### LibroDAO

La clase LibroDAO implementa el patrón DAO (Data Access Object) para encapsular el acceso a la base de datos. Esta clase proporciona métodos para realizar búsquedas avanzadas en la base de datos.

```

1 package datos;
2
3 import java.sql.*;
4
5 public class LibroDAO {
6
7     // Busca libros en la base de datos según una clave
8     public List<String> buscarLibros(String clave) {
9         List<String> resultados = new ArrayList<>();
10        String query = "SELECT * FROM libros WHERE titulo LIKE ? OR autor LIKE ? OR categoria LIKE ?";
11
12        try (Connection conn = ConexionBD.obtenerConexion();
13             PreparedStatement pstmt = conn.prepareStatement(query)) {
14
15            String busqueda = "%" + clave + "%";
16            pstmt.setString(1, busqueda);
17            pstmt.setString(2, busqueda);
18            pstmt.setString(3, busqueda);
19
20            try (ResultSet rs = pstmt.executeQuery()) {
21                while (rs.next()) {
22                    String libro = String.format("Título: %, Autor: %, Categoría: %, Precio: %.2f",
23                                                  rs.getString("titulo"),
24                                                  rs.getString("autor"),
25                                                  rs.getString("categoria"),
26                                                  rs.getDouble("precio"));
27                    resultados.add(libro);
28                }
29            }
30        } catch (SQLException e) {
31            e.printStackTrace();
32        }
33        return resultados;
34    }
35 }
36

```

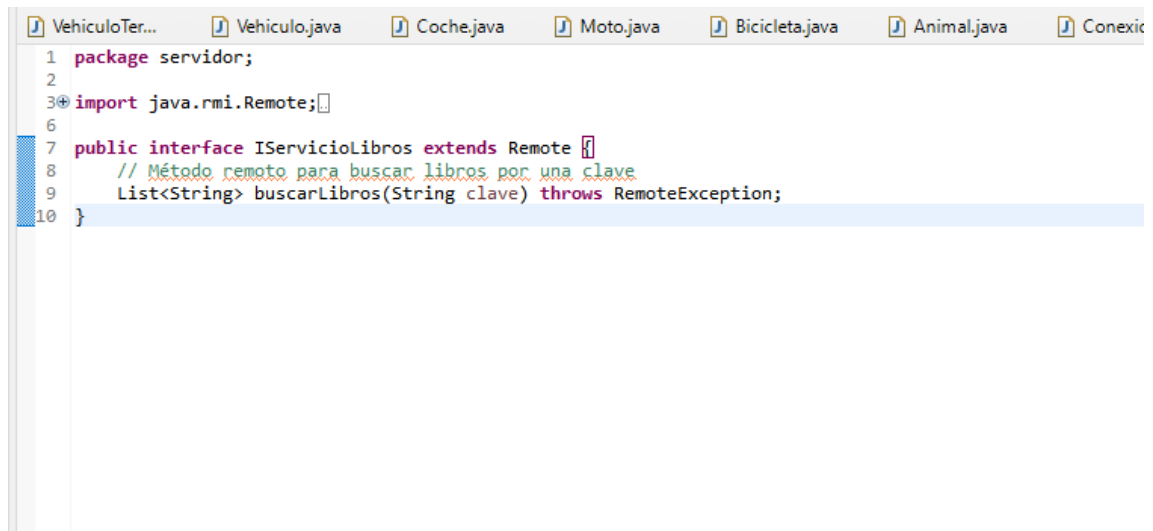
## 3. Arquitectura del Servidor

### Componentes Principales

- **ServidorLibros:** Implementa la interfaz RMI y gestiona las peticiones de los clientes.

```
ConexionBD.java LibroDAO.java ServidorLib... X IServicioLi... InterfazCli...
1 package servidor;
2
3+ import datos.LibroDAO;
7
8 public class ServidorLibros implements IServicioLibros {
9     private LibroDAO libroDAO;
10    private IServicioLibros stub;
11
12    // Constructor modificado que no extiende UnicastRemoteObject
13- public ServidorLibros() {
14        libroDAO = new LibroDAO();
15    }
16
17    // Método para exportar el objeto
18- public IServicioLibros exportar() throws RemoteException {
19        if (stub == null) {
20            stub = (IServicioLibros) UnicastRemoteObject.exportObject(this, 0);
21        }
22        return stub;
23    }
24
25    // Método para desexportar el objeto
26- public void desexportar() {
27        try {
28            if (stub != null) {
29                UnicastRemoteObject.unexportObject(this, true);
30                stub = null;
31            }
32        } catch (Exception e) {
33            // Ignorar errores de desexportación
34        }
35    }
36
37    // Implementación del método remoto para buscar libros
38- @Override
39- public List<String> buscarLibros(String clave) throws RemoteException {
40        return libroDAO.buscarLibros(clave);
41    }
42 }
```

- **IServicioLibros:** Define el contrato RMI que expone los métodos disponibles para los clientes.



```
1 package servidor;
2
3 import java.rmi.Remote;
4
5
6
7 public interface IServicioLibros extends Remote {
8     // Método remoto para buscar libros por una clave
9     List<String> buscarLibros(String clave) throws RemoteException;
10 }
```

- **ServidorManager:** Gestiona el ciclo de vida del servidor, incluyendo la inicialización y parada del mismo.



```

1 package servidor;
2
3 import java.io.BufferedReader;
4
5
6
7
8
9 public class ServidorManager {
10     private static final int PUERTO = 1099;
11     private Registry registry;
12     private ServidorLibros servidor;
13
14     // Verifica si el puerto está bloqueado
15     private boolean puertoBloqueado() {
16         try {
17             // Ejecuta un comando para verificar si el puerto está en uso
18             ProcessBuilder pb = new ProcessBuilder(
19                 "cmd", "/c", "netstat", "-ano", "|", "findstr", String.valueOf(PUERTO));
20             Process process = pb.start();
21
22             BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
23             String line = reader.readLine();
24
25             // Si se encuentra una línea, el puerto está en uso
26             return line != null && !line.isEmpty();
27         } catch (Exception e) {
28             return false;
29         }
30     }
31
32     // Libera el puerto especificado matando el proceso que lo está usando
33     private void liberarPuerto(String pid) {
34         try {
35             if (pid != null && !pid.isEmpty()) {
36                 // Ejecuta un comando para matar el proceso que está usando el puerto
37                 ProcessBuilder pb = new ProcessBuilder(
38                     "taskkill", "/F", "/PID", pid);
39                 pb.start();
40             }
41         } catch (Exception e) {
42             System.err.println("Error al liberar el puerto: " + e.getMessage());
43         }
44     }
45
46     // Inicia el servidor RMI
47     public void iniciarServidor() throws Exception {
48         try {
49             // Primero intentamos limpiar cualquier registro existente
50             try {
51                 Registry registroExistente = LocateRegistry.getRegistry(PUERTO);
52                 try {
53                     registroExistente.unbind("ServicioLibros");
54                 } catch (Exception e) {
55                     // Ignoramos si el servicio no existe
56                 }
57             } catch (Exception e) {
58                 // Ignoramos si no hay registro existente
59             }
60
61             // Esperamos un momento para asegurar que los recursos se liberan
62             Thread.sleep(100);
63         }
64     }
65 }

```

```

55         // Ignoramos si el servicio no existe
56     }
57 } catch (Exception e) {
58     // Ignoramos si no hay registro existente
59 }
60
61 // Esperamos un momento para asegurar que los recursos se liberan
62 Thread.sleep(100);
63
64 // Creamos una nueva instancia del servidor
65 servidor = new ServidorLibros();
66
67 // Intentamos crear un nuevo registro
68 try {
69     registry = LocateRegistry.createRegistry(PUERTO);
70 } catch (Exception e) {
71     // Si falla, obtenemos el registro existente
72     registry = LocateRegistry.getRegistry(PUERTO);
73 }
74
75 // Exportamos el objeto y lo vinculamos al registro
76 IServicioLibros stub = (IServicioLibros) UnicastRemoteObject.exportObject(servidor, 0);
77 registry.rebind("ServicioLibros", stub);
78
79 } catch (Exception e) {
80     // Si algo falla, intentamos limpiar todo
81     if (servidor != null) {
82         try {
83             UnicastRemoteObject.unexportObject(servidor, true);
84         } catch (Exception ex) {
85             // Ignoramos errores de limpieza
86         }
87     }
88     throw new Exception("Error al iniciar el servidor: " + e.getMessage());
89 }
90
91 // Detiene el servidor RMI
92 public void detenerServidor() throws Exception {
93     try {
94         if (registry != null) {
95             try {
96                 registry.unbind("ServicioLibros");
97             } catch (Exception e) {
98                 // Ignoramos si el servicio ya no existe
99             }
100
101             if (servidor != null) {
102                 UnicastRemoteObject.unexportObject(servidor, true);
103             }
104             UnicastRemoteObject.unexportObject(registry, true);
105             registry = null;
106             servidor = null;
107         }
108     } catch (Exception e) {
109         throw new Exception("Error al detener el servidor: " + e.getMessage());
110     }
111 }
112
113 }

```

## 4. Capacidad Multitarea

### Implementación

El servidor utiliza RMI (Remote Method Invocation) que inherentemente soporta múltiples conexiones simultáneas. Cada petición de cliente se maneja en un hilo separado, lo que permite atender múltiples clientes de manera concurrente.

### Monitorización

El servidor incluye funcionalidades para monitorizar las conexiones activas y las métricas de rendimiento, permitiendo detectar y solucionar cuellos de botella de manera eficiente.

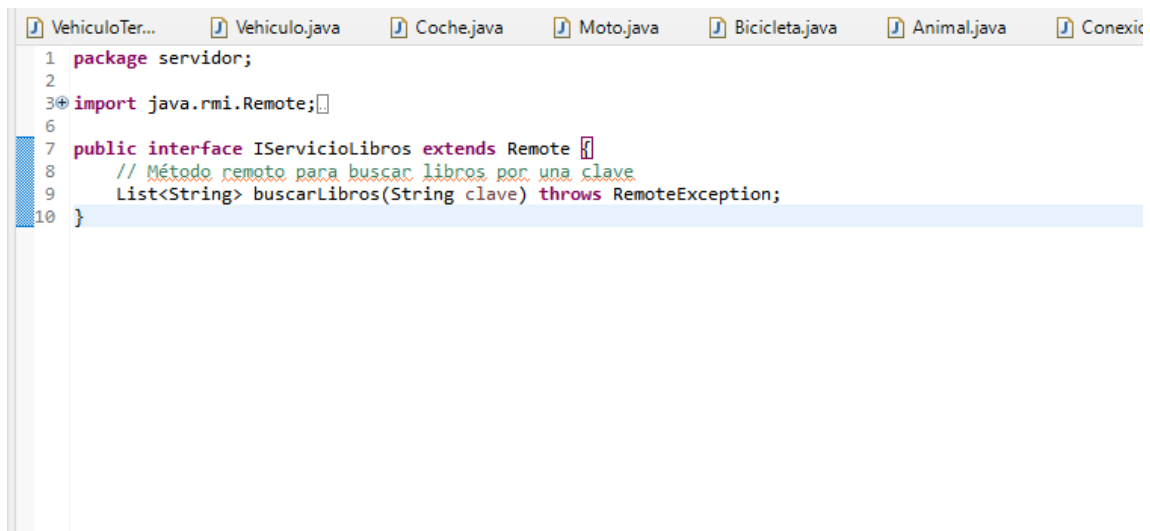
## 5. Gestión de Peticiones

## Tipos de Peticiones Soportadas

El servidor puede manejar varios tipos de peticiones, incluyendo:

- Búsqueda por título
- Búsqueda por autor
- Búsqueda por categoría
- Búsqueda por precio
- Búsqueda general

## Ejemplo de Código



```
1 package servidor;
2
3 import java.rmi.Remote;
4
5
6
7 public interface IServicioLibros extends Remote {
8     // Método remoto para buscar libros por una clave
9     List<String> buscarLibros(String clave) throws RemoteException;
10 }
```

## 6. Cliente

### Interfaz Gráfica

La clase InterfazCliente implementa una interfaz gráfica moderna y responsive utilizando Swing. La interfaz incluye:

- Panel de búsqueda con campo de texto, selector de filtros y botones de acción.
- Panel de resultados con diseño tipo card.
- Mensajes informativos para el usuario.

### Código:

```

VehiculoTer... Vehiculo.java Coche.java ConexionBD.java libreria.sql LibroDAO.java ServidorLib...
1 package interfaz;
2
3 import servidor.IServicioLibros;
4
11
12 public class InterfazCliente extends JFrame {
13     private JTextField txtBusqueda;
14     private JTextArea areaResultados;
15     private JButton btnBuscar;
16     private JButton btnLimpiar;
17     private JComboBox<String> filtroBox;
18     private IServicioLibros servicioLibros;
19     private JPanel panelResultados;
20
21 public InterfazCliente() {
22     configurarVentana();
23     inicializarComponentes();
24     btnBuscar.setEnabled(false);
25     conectarServidor();
26 }
27
28 // Conecta al servidor RMI
29 private void conectarServidor() {
30     try {
31         Registry registry = LocateRegistry.getRegistry("localhost", 1099);
32         servicioLibros = (IServicioLibros) registry.lookup("ServicioLibros");
33         btnBuscar.setEnabled(true);
34         JOptionPane.showMessageDialog(this,
35             "Conexión exitosa con el servidor",
36             "Conexión Establecida",
37             JOptionPane.INFORMATION_MESSAGE);
38     } catch (Exception e) {
39         JOptionPane.showMessageDialog(this,
40             "Error al conectar con el servidor: " + e.getMessage(),
41             "Error de Conexión",
42             JOptionPane.ERROR_MESSAGE);
43         System.exit(1);
44     }
45 }
46
47 // Configura la ventana principal
48 private void configurarVentana() {
49     setTitle("Biblioteca Virtual - Cliente");
50     setSize(1000, 700);
51     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52     setLayout(new BorderLayout(15, 15));
53     getRootPane().setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
54     getContentPane().setBackground(new Color(240, 242, 245));
55 }
56
57 // Limpia los resultados de búsqueda
58 private void limpiarResultados() {
59     txtBusqueda.setText("");
60     panelResultados.removeAll();
61     panelResultados.revalidate();
62     panelResultados.repaint();
63     filtroBox.setSelectedIndex(0);
64 }
65

```

```

VehiculoTer... Vehiculo.java Coche.java ConexionBD.java libreria.sql LibroDAO.java Servic
65
66 // Inicializa los componentes de la interfaz
67 private void inicializarComponentes() {
68     // Panel superior con título
69     JPanel panelTitulo = new JPanel();
70     panelTitulo.setLayout(new BorderLayout(panelTitulo, BorderLayout.Y_AXIS));
71     panelTitulo.setBackground(new Color(25, 118, 210));
72     panelTitulo.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0));
73
74     JLabel lblTitulo = new JLabel("Sistema de Búsqueda de Libros");
75     lblTitulo.setFont(new Font("Segoe UI", Font.BOLD, 28));
76     lblTitulo.setForeground(Color.WHITE);
77     lblTitulo.setAlignmentX(Component.CENTER_ALIGNMENT);
78
79     JLabel lblSubtitulo = new JLabel("Biblioteca Virtual");
80     lblSubtitulo.setFont(new Font("Segoe UI Light", Font.PLAIN, 16));
81     lblSubtitulo.setForeground(new Color(224, 224, 224));
82     lblSubtitulo.setAlignmentX(Component.CENTER_ALIGNMENT);
83
84     panelTitulo.add(lblTitulo);
85     panelTitulo.add(Box.createRigidArea(new Dimension(0, 5)));
86     panelTitulo.add(lblSubtitulo);
87
88     // Panel de búsqueda
89     JPanel panelBusqueda = new JPanel(new GridBagLayout());
90     panelBusqueda.setBackground(Color.WHITE);
91     panelBusqueda.setBorder(BorderFactory.createCompoundBorder(
92         BorderFactory.createLineBorder(new Color(200, 200, 200), 1),
93         BorderFactory.createEmptyBorder(20, 20, 20, 20)
94     ));
95
96     GridBagConstraints gbc = new GridBagConstraints();
97     gbc.fill = GridBagConstraints.HORIZONTAL;
98     gbc.insets = new Insets(5, 5, 5, 5);
99
100    // Campo de búsqueda con icono
101    JPanel searchPanel = new JPanel(new BorderLayout(5, 0));
102    searchPanel.setBackground(Color.WHITE);
103    txtBusqueda = new JTextField(30);
104    txtBusqueda.setFont(new Font("Segoe UI", Font.PLAIN, 14));
105    txtBusqueda.setBorder(BorderFactory.createCompoundBorder(
106        BorderFactory.createLineBorder(new Color(200, 200, 200)),
107        BorderFactory.createEmptyBorder(8, 10, 8, 10)
108    ));
109    txtBusqueda.addActionListener(e -> buscarLibros());
110
111    // ComboBox personalizado
112    String[] opciones = {"Todos", "Título", "Autor", "Precio"};
113    filtroBox = new JComboBox<>(opciones);
114    filtroBox.setFont(new Font("Segoe UI", Font.PLAIN, 14));
115    filtroBox.setPreferredSize(new Dimension(150, 38));
116    filtroBox.setBackground(Color.WHITE);
117
118    // Botones modernos
119    btnBuscar = new JButton("Buscar");
120    btnLimpiar = new JButton("Limpiar");
121

```

```

VehiculoTer...  Vehiculo.java  Coche.java  ConexionBD.java  libreria.sql  LibroDAO.java  S
122      // Eventos
123      btnBuscar.addActionListener(e -> buscarLibros());
124      btnLimpiar.addActionListener(e -> limpiarResultados());
125
126      // Estilo botones
127      configurarBoton(btnBuscar, new Color(25, 118, 210));
128      configurarBoton(btnLimpiar, new Color(158, 158, 158));
129
130      // Layout
131      gbc.gridx = 0; gbc.gridy = 0;
132      panelBusqueda.add(new JLabel("Buscar:"), gbc);
133
134      gbc.gridx = 1; gbc.weightx = 1.0;
135      panelBusqueda.add(txtBusqueda, gbc);
136
137      gbc.gridx = 2; gbc.weightx = 0;
138      panelBusqueda.add(new JLabel("Filtrar por:"), gbc);
139
140      gbc.gridx = 3;
141      panelBusqueda.add(filtroBox, gbc);
142
143      gbc.gridx = 4;
144      panelBusqueda.add(btnBuscar, gbc);
145
146      gbc.gridx = 5;
147      panelBusqueda.add(btnLimpiar, gbc);
148
149      // Área de resultados
150      panelResultados = new JPanel();
151      panelResultados.setLayout(new GridLayout(0, 2, 10, 10)); // 2 columnas, espaciado 10px
152      panelResultados.setBackground(new Color(240, 242, 245));
153
154      JScrollPane scrollResultados = new JScrollPane(panelResultados);
155      scrollResultados.setBorder(BorderFactory.createCompoundBorder(
156          BorderFactory.createTitledBorder(null, "Resultados",
157              TitledBorder.DEFAULT_JUSTIFICATION,
158              TitledBorder.DEFAULT_POSITION,
159              new Font("Segoe UI", Font.BOLD, 14)),
160          BorderFactory.createEmptyBorder(10, 10, 10, 10)
161      ));
162      scrollResultados.setPreferredSize(new Dimension(0, 400));
163
164      // Panel principal
165      JPanel mainPanel = new JPanel(new BorderLayout(15, 15));
166      mainPanel.setBackground(new Color(240, 242, 245));
167      mainPanel.add(panelTitulo, BorderLayout.NORTH);
168      mainPanel.add(panelBusqueda, BorderLayout.CENTER);
169      mainPanel.add(scrollResultados, BorderLayout.SOUTH);
170
171      add(mainPanel);
172  }

```

```

173
174 // Añadir método auxiliar para configurar botones
175 private void configurarBoton(JButton btn, Color color) {
176     btn.setFont(new Font("Segoe UI", Font.BOLD, 14));
177     btn.setPreferredSize(new Dimension(120, 38));
178     btn.setBackground(color);
179     btn.setForeground(Color.WHITE);
180     btn.setFocusPainted(false);
181     btn.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));
182     btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
183
184     btn.addMouseListener(new java.awt.event.MouseAdapter() {
185         public void mouseEntered(java.awt.event.MouseEvent evt) {
186             btn.setBackground(color.brighter());
187         }
188         public void mouseExited(java.awt.event.MouseEvent evt) {
189             btn.setBackground(color);
190         }
191     });
192 }
193
194 // Realiza la búsqueda de libros
195 private void buscarLibros() {
196     String termino = txtBusqueda.getText().trim();
197     String filtro = (String) filtroBox.getSelectedItem();
198
199     if (termino.isEmpty()) {
200         JOptionPane.showMessageDialog(this,
201             "Por favor ingrese un término de búsqueda",
202             "Advertencia",
203             JOptionPane.WARNING_MESSAGE);
204         return;
205     }
206
207     try {
208         List<String> resultados = serviciolibros.buscarLibros(termino);
209         mostrarResultadosFiltrados(resultados, filtro);
210     } catch (Exception e) {
211         JOptionPane.showMessageDialog(this,
212             "Error al buscar libros: " + e.getMessage(),
213             "Error",
214             JOptionPane.ERROR_MESSAGE);
215     }
216 }
217
218 // Muestra los resultados filtrados en la interfaz
219 private void mostrarResultadosFiltrados(List<String> resultados, String filtro) {
220     panelResultados.removeAll();
221
222     if (resultados.isEmpty()) {
223         JLabel noResultados = new JLabel("No se encontraron libros.");
224         noResultados.setFont(new Font("Segoe UI", Font.PLAIN, 14));
225         panelResultados.add(noResultados);
226         panelResultados.revalidate();
227         panelResultados.repaint();
228         return;
229     }
230

```

```

VehiculoTer...  Vehiculo.java  Coche.java  ConexionBD.java  libreria.sql  LibroDAO.java  Servidor
230
231     for (String libro : resultados) {
232         JPanel card = crearCard(libro, filtro);
233         if (card != null) {
234             panelResultados.add(card);
235         }
236     }
237
238     panelResultados.revalidate();
239     panelResultados.repaint();
240 }
241
242 // Crea una tarjeta para mostrar la información de un libro
243 private JPanel crearCard(String libro, String filtro) {
244     JPanel card = new JPanel();
245     card.setLayout(new BoxLayout(card, BoxLayout.Y_AXIS));
246     card.setBackground(Color.WHITE);
247     card.setBorder(BorderFactory.createCompoundBorder(
248         BorderFactory.createLineBorder(new Color(200, 200, 200), 1),
249         BorderFactory.createEmptyBorder(15, 15, 15, 15)
250     ));
251
252     // Cambiar el split para usar una expresión regular que mantenga el precio completo
253     String[] partes = libro.split("(?<=\\D),|(?=\\D)");
254     boolean mostrarCard = false;
255
256     switch (filtro) {
257         case "Todos":
258             for (String parte : partes) {
259                 agregarCampoACard(card, parte.trim());
260             }
261             mostrarCard = true;
262             break;
263
264         case "Título":
265             for (String parte : partes) {
266                 if (parte.trim().startsWith("Título:")) {
267                     agregarCampoACard(card, parte.trim());
268                     mostrarCard = true;
269                 }
270             }
271             break;
272
273         case "Autor":
274             for (String parte : partes) {
275                 if (parte.trim().startsWith("Autor:")) {
276                     agregarCampoACard(card, parte.trim());
277                     mostrarCard = true;
278                 }
279             }
280             break;
281
282         case "Precio":
283             for (String parte : partes) {
284                 if (parte.trim().startsWith("Precio:")) {
285                     agregarCampoACard(card, parte.trim());
286                     mostrarCard = true;
287                 }
288             }

```



```

266         if (parte.trim().startsWith("Título:")) {
267             agregarCampoACard(card, parte.trim());
268             mostrarCard = true;
269         }
270     }
271     break;
272
273     case "Autor":
274         for (String parte : partes) {
275             if (parte.trim().startsWith("Autor:")) {
276                 agregarCampoACard(card, parte.trim());
277                 mostrarCard = true;
278             }
279         }
280         break;
281
282     case "Precio":
283         for (String parte : partes) {
284             if (parte.trim().startsWith("Precio:")) {
285                 agregarCampoACard(card, parte.trim());
286                 mostrarCard = true;
287             }
288         }
289         break;
290     }
291
292     return mostrarCard ? card : null;
293 }
294
295 // Añade un campo a la tarjeta
296 private void agregarCampoACard(JPanel card, String texto) {
297     JLabel label = new JLabel(texto);
298     label.setFont(new Font("Segoe UI", Font.PLAIN, 14));
299     label.setAlignmentX(Component.LEFT_ALIGNMENT);
300     label.setBorder(BorderFactory.createEmptyBorder(2, 0, 2, 0));
301
302     // Aplicar estilos según el tipo de información
303     if (texto.startsWith("Título:")) {
304         label.setFont(new Font("Segoe UI", Font.BOLD, 16));
305         label.setForeground(new Color(25, 118, 210));
306     } else if (texto.startsWith("Autor:")) {
307         label.setForeground(new Color(0, 100, 0));
308     } else if (texto.startsWith("Precio:")) {
309         label.setForeground(new Color(178, 34, 34));
310         label.setFont(new Font("Segoe UI", Font.BOLD, 14));
311     }
312
313     card.add(label);
314     card.add(Box.createRigidArea(new Dimension(0, 5)));
315 }
316
317 public static void main(String[] args) {
318     SwingUtilities.invokeLater(() -> {
319         InterfazCliente cliente = new InterfazCliente();
320         cliente.setLocationRelativeTo(null);
321         cliente.setVisible(true);
322     });
323 }
324 }

```

## 7. Servidor

### Interfaz Gráfica

La clase InterfazServidor implementa una interfaz gráfica moderna y responsive utilizando Swing. La interfaz incluye:

- Panel de control con botones para iniciar y detener el servidor.

- Indicadores de estado para mostrar el estado del servidor y la conexión a la base de datos.
- Área de logs para mostrar mensajes informativos y eventos del servidor.

## Código:

```

VehiculoTer...  Vehiculo.java  Coche.java  Moto.java  Bicicleta.java  Animal.java  Mamifero...
1  package interfaz;
2
3  import java.awt.*;
12
13  public class InterfazServidor extends JFrame {
14      private JButton btnIniciar;
15      private JButton btnDetener;
16      private JTextArea areaLogs;
17      private JLabel lblEstado;
18      private JLabel lblConexion;
19      private Registry registry;
20      private IServicioLibros stub;
21      private ServidorLibros servicioLibros;
22      private boolean servidorIniciado = false;
23
24  public InterfazServidor() {
25      // Configura la ventana principal
26      configurarVentana();
27      // Inicializa los componentes de la interfaz
28      inicializarComponentes();
29      // Configura los eventos de los botones
30      configurarEventos();
31      // Verifica el estado del servidor al iniciar
32      verificarEstadoServidor();
33  }
34
35  // Añadir nuevo método para verificación
36  private void verificarEstadoServidor() {
37      try {
38          // Intenta conectarse al registro RMI en el puerto 1099
39          Registry testRegistry = LocateRegistry.getRegistry(1099);
40          try {
41              // Intenta buscar el servicio
42              testRegistry.lookup("ServicioLibros");
43              // Si no lanza excepción, el servidor está activo
44              servidorIniciado = true;
45              btnIniciar.setEnabled(false);
46              btnDetener.setEnabled(true);
47              lblEstado.setText("Estado: Ejecutando");
48              lblEstado.setForeground(new Color(46, 139, 87));
49              lblConexion.setText("Base de datos: Conectada");
50              lblConexion.setForeground(new Color(46, 139, 87));
51              agregarLog("Servidor detectado en ejecución");
52          } catch (Exception e) {
53              // El registro existe pero el servicio no está disponible
54              agregarLog("Puerto 1099 en uso pero el servicio no está disponible");
55          }
56      } catch (Exception e) {
57          // El registro no existe, el servidor está completamente detenido
58          servidorIniciado = false;
59          agregarLog("Servidor no detectado");
60      }
61  }

```

```

VehiculoTer...  Vehiculo.java  Coche.java  Moto.java  Bicicleta.java  Animal.java  Mamifer
63 // Modificar el método iniciarServidor()
64 private void iniciarServidor() throws Exception {
65     if (servidorIniciado) {
66         agregarLog("Aviso: El servidor ya está en ejecución");
67         JOptionPane.showMessageDialog(this,
68             "El servidor ya está en ejecución",
69             "Aviso",
70             JOptionPane.INFORMATION_MESSAGE);
71         return;
72     }
73
74     try {
75         // Primero intentamos limpiar cualquier instancia anterior
76         try {
77             Registry registroExistente = LocateRegistry.getRegistry(1099);
78             try {
79                 registroExistente.unbind("ServicioLibros");
80             } catch (Exception e) {
81                 // Ignoramos si el servicio no existe
82             }
83         } catch (Exception e) {
84             // Ignoramos si no hay registro existente
85         }
86
87         // Esperamos un momento para asegurar que los recursos se liberan
88         Thread.sleep(100);
89
90         // Creamos una nueva instancia del servicio
91         servicioLibros = new ServidorLibros();
92
93         // Intentamos crear un nuevo registro
94         try {
95             registry = LocateRegistry.createRegistry(1099);
96         } catch (Exception e) {
97             // Si falla, obtenemos el registro existente
98             registry = LocateRegistry.getRegistry(1099);
99         }
100
101         // Exportamos el objeto y lo vinculamos al registro
102         stub = (IServicioLibros) UnicastRemoteObject.exportObject(servicioLibros, 0);
103         registry.rebind("ServicioLibros", stub);
104
105         servidorIniciado = true;
106         lblConexion.setText("Base de datos: Conectada");
107         lblConexion.setForeground(new Color(46, 139, 87));
108         agregarLog("Servidor iniciado correctamente");
109
110     } catch (Exception e) {
111         // Si algo falla, intentamos limpiar todo
112         if (servicioLibros != null) {
113             try {
114                 UnicastRemoteObject.unexportObject(servicioLibros, true);
115             } catch (Exception ex) {
116                 // Ignoramos errores de limpieza
117             }
118         }
119         throw new Exception(e.getMessage());
120     }
121 }

```

```

VehiculoTer... Vehiculo.java Coche.java Moto.java Bicicleta.java Animal.java Mamifero.java
123 // Modificar el método detenerServidor()
124 private void detenerServidor() throws Exception {
125     try {
126         if (!servidorIniciado) {
127             throw new Exception("El servidor no está en ejecución");
128         }
129         if (registry != null) {
130             // Desvincula el objeto del servicio del registro
131             registry.unbind("ServicioLibros");
132             if (stub != null) {
133                 // Desexporta el objeto del servicio
134                 UnicastRemoteObject.unexportObject(servicioLibros, true);
135             }
136             // Desexporta el registro
137             UnicastRemoteObject.unexportObject(registry, true);
138             registry = null;
139             stub = null;
140             servicioLibros = null;
141             servidorIniciado = false;
142             lblConexion.setText("Base de datos: No conectada");
143             lblConexion.setForeground(Color.RED);
144         }
145     } catch (Exception e) {
146         throw new Exception("Error al detener el servidor: " + e.getMessage());
147     }
148 }
149
150 // Modificar la parte relevante de configurarEventos()
151 private void configurarEventos() {
152     btnIniciar.addActionListener(e -> {
153         try {
154             iniciarServidor();
155             // Solo actualizar la UI si el servidor se inició correctamente
156             if (servidorIniciado) {
157                 btnIniciar.setEnabled(false);
158                 btnDetener.setEnabled(true);
159                 lblEstado.setText("Estado: Ejecutando");
160                 lblEstado.setForeground(new Color(46, 139, 87));
161                 agregarLog("Servidor iniciado correctamente");
162             }
163         } catch (Exception ex) {
164             agregarLog("Error al iniciar el servidor: " + ex.getMessage());
165             JOptionPane.showMessageDialog(this,
166                 "Error al iniciar el servidor: " + ex.getMessage(),
167                 "Error",
168                 JOptionPane.ERROR_MESSAGE);
169         }
170     });

```

```

171
172     btnDetener.addActionListener(e -> {
173         try {
174             detenerServidor();
175             btnIniciar.setEnabled(true);
176             btnDetener.setEnabled(false);
177             lblEstado.setText("Estado: Detenido");
178             lblEstado.setForeground(Color.RED);
179             agregarLog("Servidor detenido correctamente");
180         } catch (Exception ex) {
181             agregarLog("Error al detener el servidor: " + ex.getMessage());
182             if (!ex.getMessage().contains("no está en ejecución")) {
183                 JOptionPane.showMessageDialog(this,
184                     "Error al detener el servidor: " + ex.getMessage(),
185                     "Error",
186                     JOptionPane.ERROR_MESSAGE);
187             }
188         }
189     });
190 }
191
192 private void agregarLog(String mensaje) {
193     String timestamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
194     areaLogs.append(timestamp + " - " + mensaje + "\n");
195     // Auto-scroll al final
196     areaLogs.setCaretPosition(areaLogs.getDocument().getLength());
197 }
198
199 private void configurarVentana() {
200     setTitle("Panel de Control - Servidor de Biblioteca");
201     setSize(800, 600);
202     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
203     setLayout(new BorderLayout(10, 10));
204     getRootPane().setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
205 }
206
207 private void inicializarComponentes() {
208     // Panel de título
209     JPanel panelTitulo = new JPanel(new FlowLayout(FlowLayout.CENTER));
210     JLabel titulo = new JLabel("Control del Servidor de Biblioteca");
211     titulo.setFont(new Font("Arial", Font.BOLD, 24));
212     panelTitulo.add(titulo);
213
214     // Panel de control
215     JPanel panelControl = new JPanel(new GridBagLayout());
216     panelControl.setBorder(BorderFactory.createTitledBorder("Panel de Control"));
217     GridBagConstraints gbc = new GridBagConstraints();
218
219     // Botones
220     btnIniciar = new JButton("Iniciar Servidor");
221     btnDetener = new JButton("Detener Servidor");
222     btnIniciar.setPreferredSize(new Dimension(150, 40));
223     btnDetener.setPreferredSize(new Dimension(150, 40));
224     btnDetener.setEnabled(false);

```

```

217 // debugConstraints gbc = new GridBagConstraints();
218
219 // Botones
220 btnIniciar = new JButton("Iniciar Servidor");
221 btnDetener = new JButton("Detener Servidor");
222 btnIniciar.setPreferredSize(new Dimension(150, 40));
223 btnDetener.setPreferredSize(new Dimension(150, 40));
224 btnDetener.setEnabled(false);
225
226 // Estilo botones
227 btnIniciar.setBackground(new Color(46, 139, 87));
228 btnIniciar.setForeground(Color.WHITE);
229 btnDetener.setBackground(new Color(178, 34, 34));
230 btnDetener.setForeground(Color.WHITE);
231
232 // Estado y conexión
233 lblEstado = new JLabel("Estado: Detenido");
234 lblEstado.setFont(new Font("Arial", Font.BOLD, 14));
235 lblConexion = new JLabel("Base de datos: No conectada");
236 lblConexion.setFont(new Font("Arial", Font.PLAIN, 14));
237
238 // Configurar layout
239 gbc.gridx = 0; gbc.gridy = 0;
240 gbc.insets = new Insets(5,5,5,5);
241 panelControl.add(btnIniciar, gbc);
242
243 gbc.gridx = 1;
244 panelControl.add(btnDetener, gbc);
245
246 gbc.gridx = 0; gbc.gridy = 1;
247 gbc.gridwidth = 2;
248 panelControl.add(lblEstado, gbc);
249
250 gbc.gridy = 2;
251 panelControl.add(lblConexion, gbc);
252
253 // Área de logs
254 areaLogs = new JTextArea();
255 areaLogs.setEditable(false);
256 areaLogs.setFont(new Font("Monospaced", Font.PLAIN, 12));
257 JScrollPane scrollLogs = new JScrollPane(areaLogs);
258 scrollLogs.setBorder(BorderFactory.createTitledBorder("Registro de Eventos"));
259
260 // Panel principal
261 JPanel mainPanel = new JPanel(new BorderLayout(10, 10));
262 mainPanel.add(panelTitulo, BorderLayout.NORTH);
263 mainPanel.add(panelControl, BorderLayout.CENTER);
264 mainPanel.add(scrollLogs, BorderLayout.SOUTH);
265
266 add(mainPanel);
267 }
268
269 public static void main(String[] args) {
270     SwingUtilities.invokeLater(() -> {
271         InterfazServidor servidor = new InterfazServidor();
272         servidor.setLocationRelativeTo(null);
273         servidor.setVisible(true);
274     });
275 }
276 }

```

## 8. Documentación Visual

### Capturas de Pantalla

El documento PDF incluye capturas de pantalla que demuestran el funcionamiento de la aplicación. Estas capturas muestran:

## Pantalla de inicio.

### INTERFAZ SERVIDOR



### SERVIDOR INICIADO

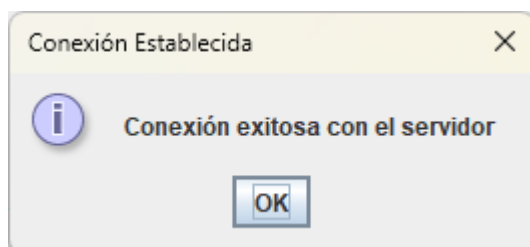


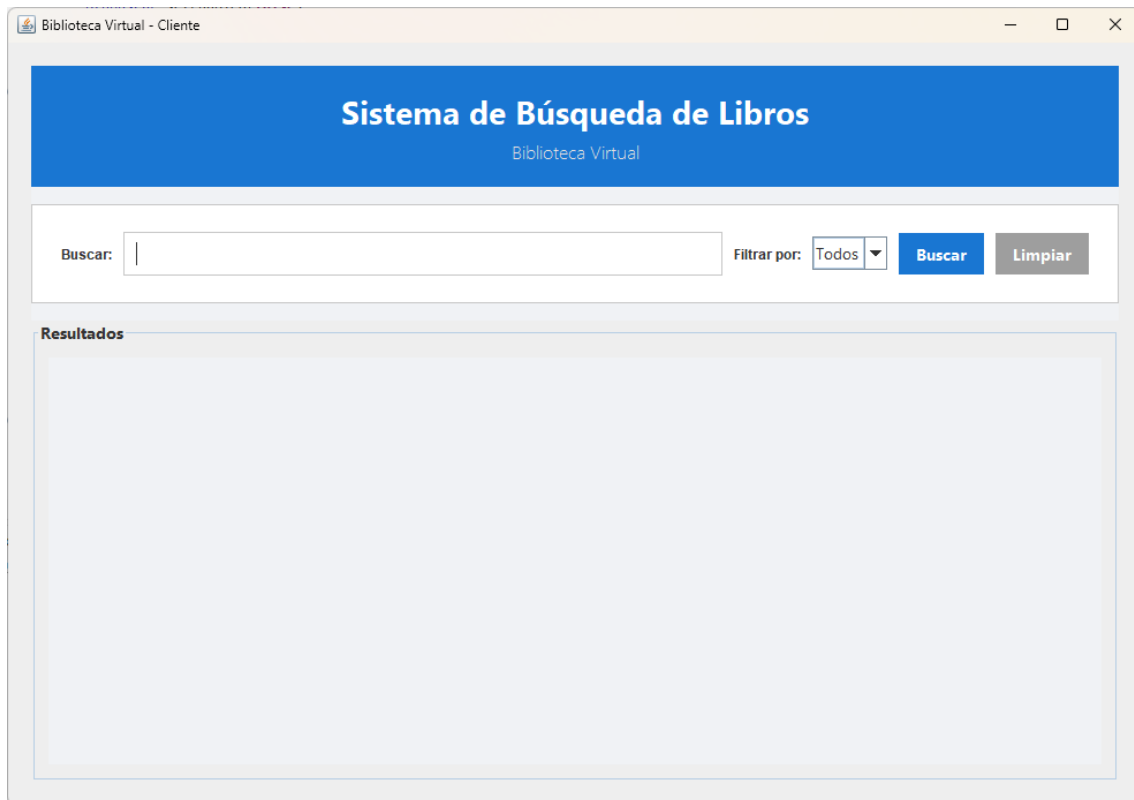
## SERVIDOR DETENIDO





## INTERFAZ CLIENTE



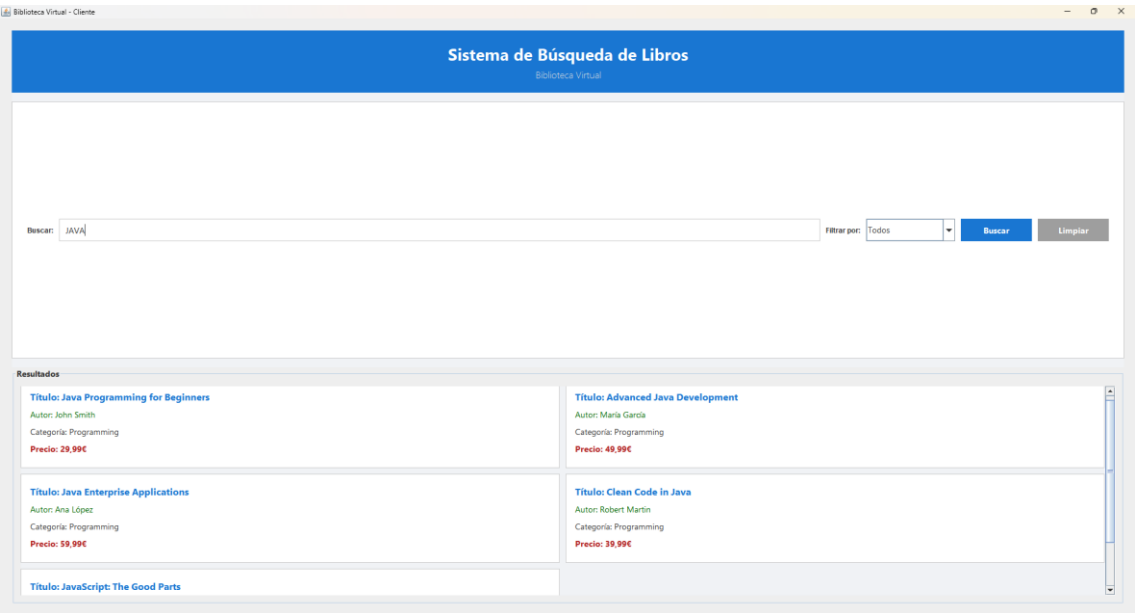


## Proceso de búsqueda.

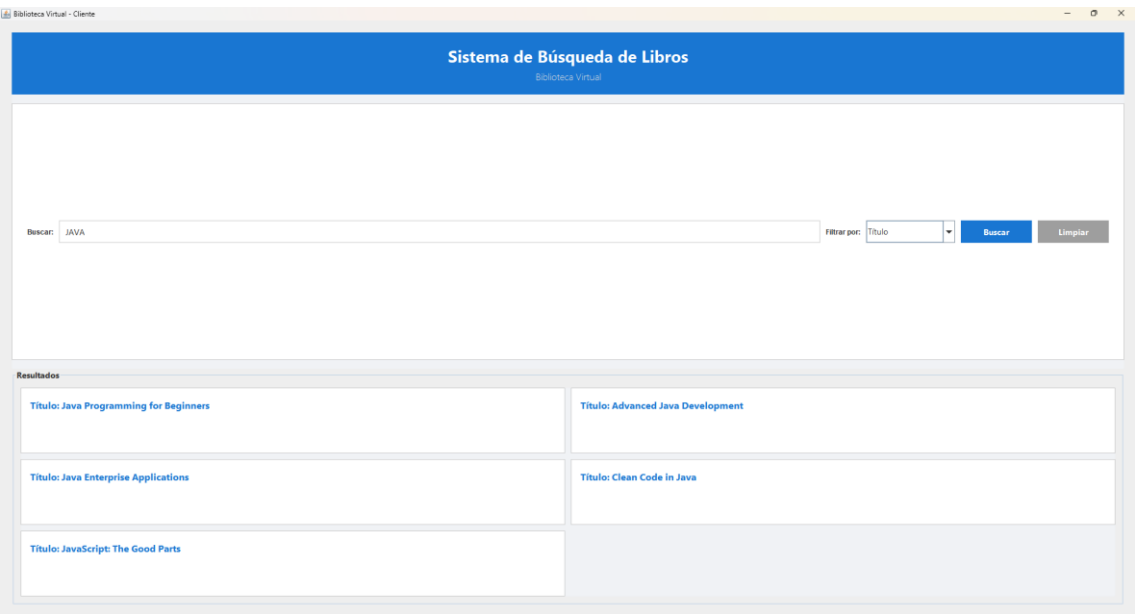


# Resultados filtrados.

# TODOS



# TITULO



# AUTOR

Biblioteca Virtual - Cliente

## Sistema de Búsqueda de Libros

Biblioteca Virtual

Buscar:  Filtrar por:

Resultados

|                          |                      |
|--------------------------|----------------------|
| Autor: John Smith        | Autor: María García  |
| Autor: Ana López         | Autor: Robert Martin |
| Autor: Douglas Crockford |                      |

## PRECIO

Biblioteca Virtual - Cliente

## Sistema de Búsqueda de Libros

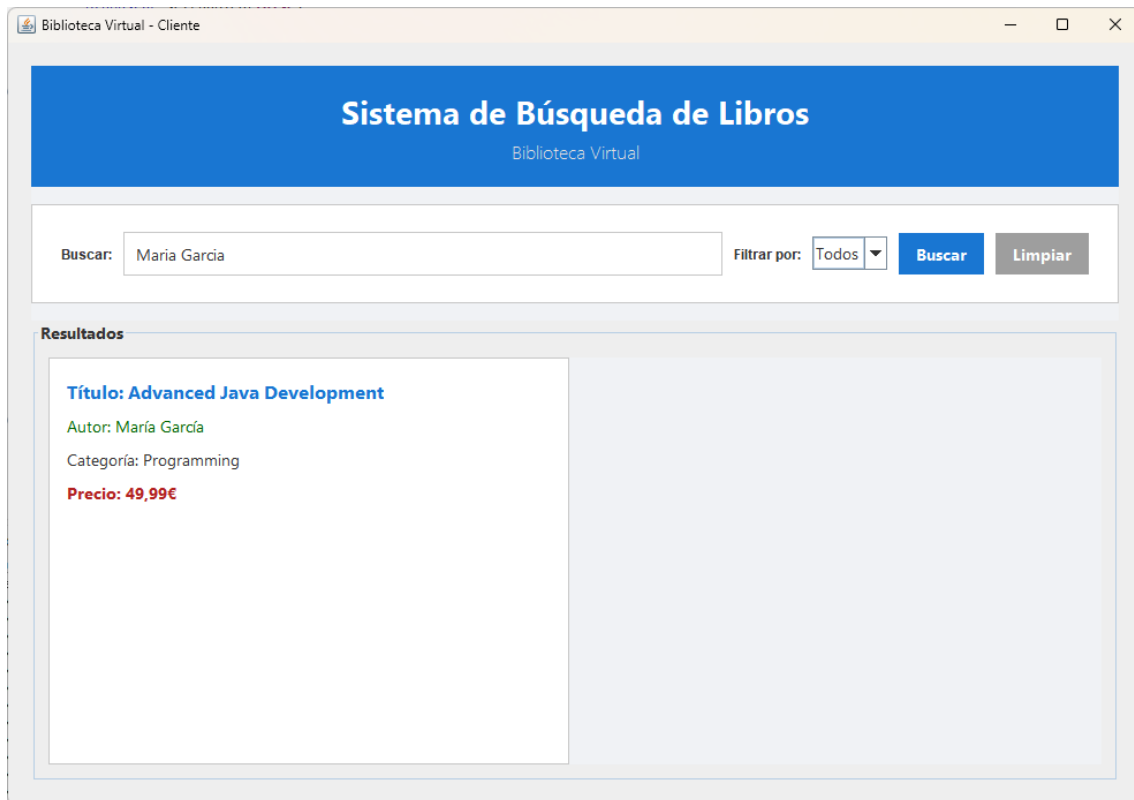
Biblioteca Virtual

Buscar:  Filtrar por:

Resultados

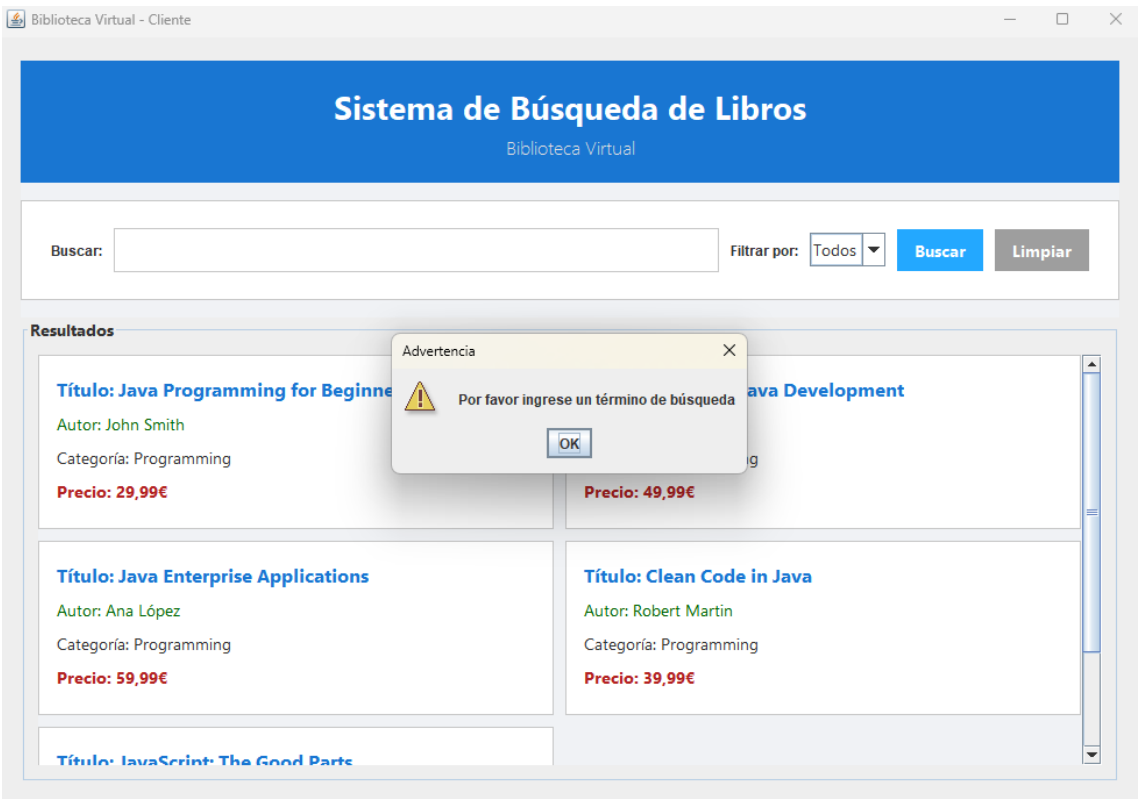
|                |                |
|----------------|----------------|
| Precio: 29,99€ | Precio: 49,99€ |
| Precio: 59,99€ | Precio: 39,99€ |
| Precio: 34,99€ |                |

## COMPROBACION DE UTILIZACION POR OTROS FILTROS

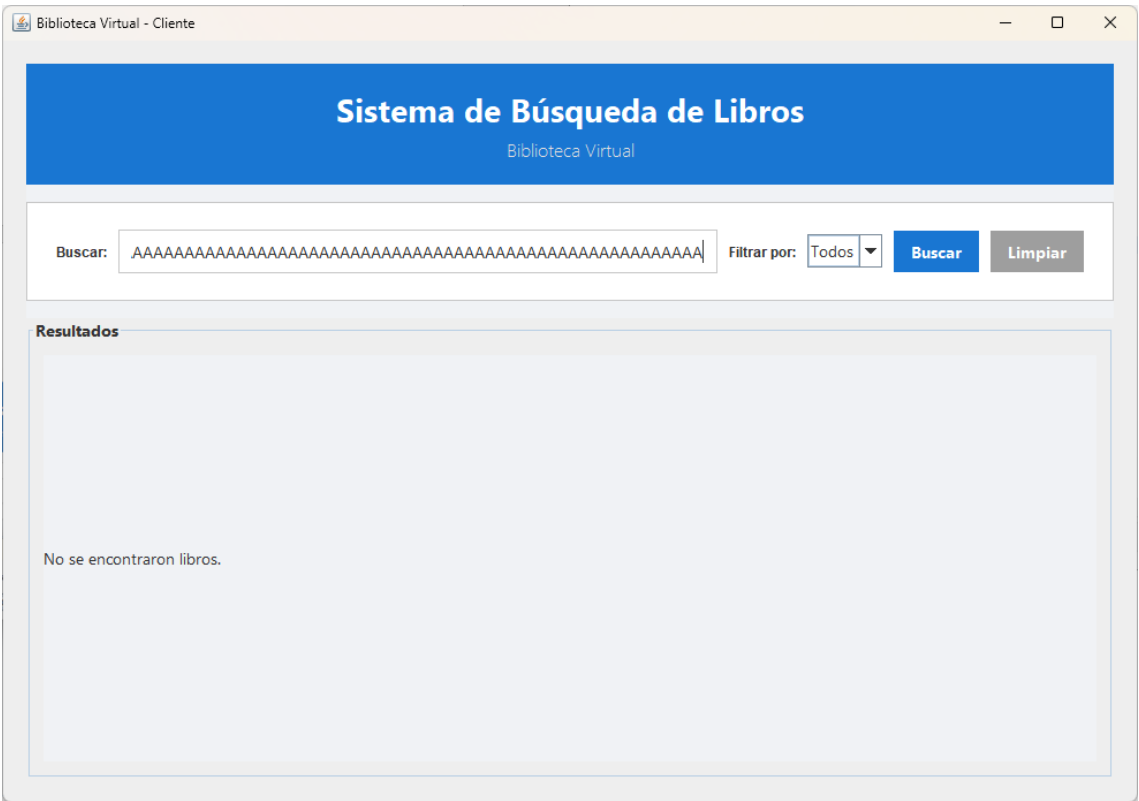


**Gestión de errores.**

SINO SE BUSCA NADA

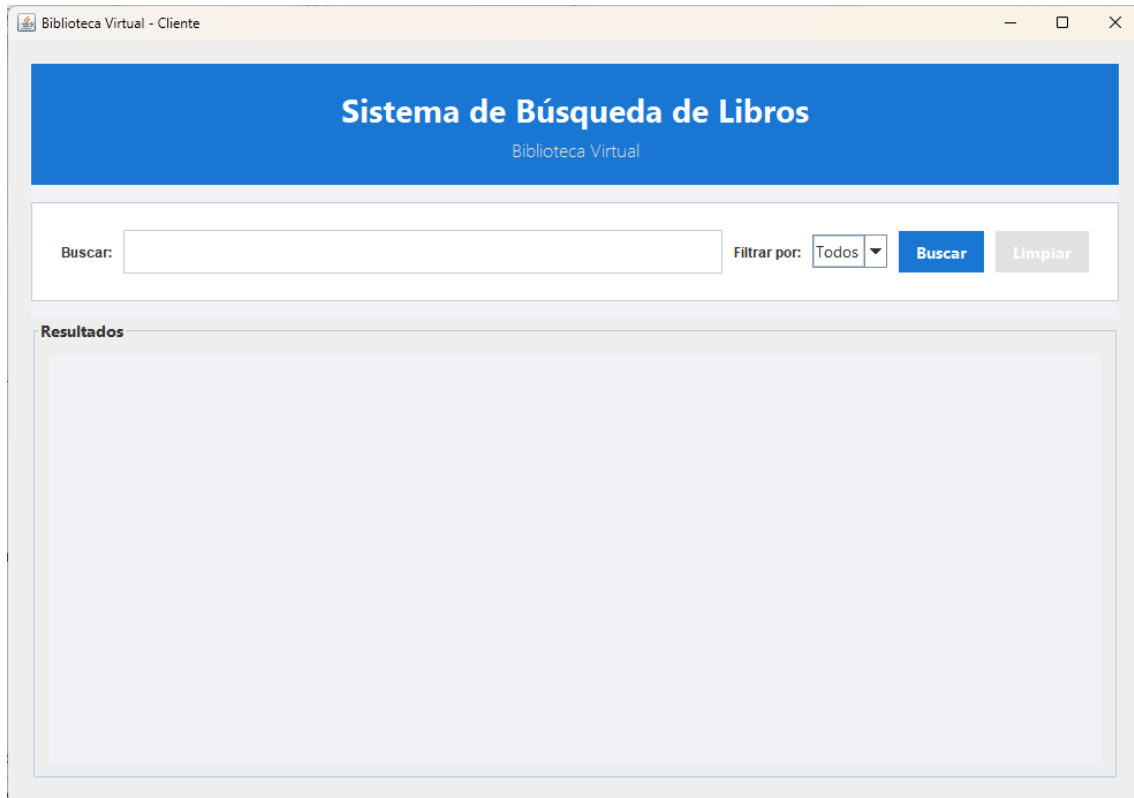


SI SE BUSCA ALGO QUE NO ESTA EN LA BDD



## FUNCIONALIDAD

SI SE LE DA AL BOTON LIMPIAR SE LIMPIA TANTO LOS RESULTADOS COMO EL CAMPO DE BUSQUEDA



## Webgrafía

Stack Overflow. (n.d.). Stack Overflow. Recuperado el 18 de noviembre de 2024, de <https://stackoverflow.com/questions/11235827/eclipse-error-could-not-find-or-load-main-class>

## ANEXO I

└─ cliente/

│ │ └─ ClienteLibros.java

```
// ClienteLibros.java
```

```
package cliente;
```

```

import servidor.IServicioLibros;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.util.List;

import java.util.Scanner;


public class ClienteLibros {

    public static void main(String[] args) {

        try {

            Registry registry = LocateRegistry.getRegistry("localhost", 1099);

            IServicioLibros servicioLibros = (IServicioLibros)
registry.lookup("ServicioLibros");


            Scanner scanner = new Scanner(System.in);

            while (true) {

                System.out.print("Ingrese término de búsqueda (o 'salir' para terminar): ");

                String clave = scanner.nextLine();

                if (clave.equalsIgnoreCase("salir")) break;

                List<String> resultados = servicioLibros.buscarLibros(clave);

                if (resultados.isEmpty()) {

                    System.out.println("No se encontraron libros.");

                } else {

                    System.out.println("Libros encontrados:");

```



```

        resultados.forEach(System.out::println);
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

└─ **datos/**

├─├─└─ **ConexionBD.java**

```
package datos;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
public class ConexionBD {
```

```
    private static final String URL = "jdbc:mysql://localhost:3306/libreria";
```

```
    private static final String USER = "root";
```

```
    private static final String PASSWORD = "Tcachuk93";
```

```
    static {
```

```
        try {
```

```
            // Registrar el driver explícitamente
```

```
            Class.forName("com.mysql.cj.jdbc.Driver");
```

```

        System.out.println("Driver MySQL registrado correctamente");
    } catch (ClassNotFoundException e) {

        System.err.println("Error al registrar el driver MySQL: " + e.getMessage());

        throw new RuntimeException("No se pudo registrar el driver MySQL", e);

    }
}

// Obtiene una conexión a la base de datos

public static Connection obtenerConexion() throws SQLException {

    try {

        Connection conexion = DriverManager.getConnection(URL, USER,
PASSWORD);

        System.out.println("Conexión establecida con éxito a la base de datos");

        return conexion;

    } catch (SQLException e) {

        System.err.println("Error al conectar a la base de datos: " + e.getMessage());

        throw e;

    }

}
}

```

**LibroDAO.java**

```
package datos;
```

```
import java.sql.*;
```

```
import java.util.*;
```

```

public class LibroDAO {

    // Busca libros en la base de datos según una clave

    public List<String> buscarLibros(String clave) {

        List<String> resultados = new ArrayList<>();

        String query = "SELECT * FROM libros WHERE titulo LIKE ? OR autor LIKE ?
OR categoria LIKE ?";

        try (Connection conn = ConexionBD.obtenerConexion();

            PreparedStatement pstmt = conn.prepareStatement(query)) {

            String busqueda = "%" + clave + "%";

            pstmt.setString(1, busqueda);

            pstmt.setString(2, busqueda);

            pstmt.setString(3, busqueda);

            try (ResultSet rs = pstmt.executeQuery()) {

                while (rs.next()) {

                    String libro = String.format("Título: %s, Autor: %s, Categoría: %s, Precio:
%.2f€",

                        rs.getString("titulo"),

                        rs.getString("autor"),

                        rs.getString("categoria"),

                        rs.getDouble("precio"));

                    resultados.add(libro);

                }
            }
        }
    }
}

```

```

        }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return resultados;

    }

}

|  |—— interfaz/
|  |  |—— InterfazCliente.java

```

```
package interfaz;
```

```
import servidor.IServicioLibros;
```

```
import javax.swing.*;
```

```
import javax.swing.border.TitledBorder;
```

```
import java.awt.*;
```

```
import java.rmi.registry.LocateRegistry;
```

```
import java.rmi.registry.Registry;
```

```
import java.util.List;
```

```
public class InterfazCliente extends JFrame {
```

```
    private JTextField txtBusqueda;
```

```
    private JTextArea areaResultados;
```

```
    private JButton btnBuscar;
```

```
    private JButton btnLimpiar;
```

```

private JComboBox<String> filtroBox;

private IServicioLibros servicioLibros;

private JPanel panelResultados;


public InterfazCliente() {

    configurarVentana();

    inicializarComponentes();

    btnBuscar.setEnabled(false);

    conectarServidor();

}


// Conecta al servidor RMI

private void conectarServidor() {

    try {

        Registry registry = LocateRegistry.getRegistry("localhost", 1099);

        servicioLibros = (IServicioLibros) registry.lookup("ServicioLibros");

        btnBuscar.setEnabled(true);

        JOptionPane.showMessageDialog(this,

            "Conexión exitosa con el servidor",

            "Conexión Establecida",

            JOptionPane.INFORMATION_MESSAGE);

    } catch (Exception e) {

        JOptionPane.showMessageDialog(this,

            "Error al conectar con el servidor: " + e.getMessage(),

            "Error de Conexión",

```

```

        JOptionPane.ERROR_MESSAGE);

        System.exit(1);

    }

}

// Configura la ventana principal

private void configurarVentana() {

    setTitle("Biblioteca Virtual - Cliente");

    setSize(1000, 700);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setLayout(new BorderLayout(15, 15));

    getRootPane().setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

    getContentPane().setBackground(new Color(240, 242, 245));

}

// Limpia los resultados de búsqueda

private void limpiarResultados() {

    txtBusqueda.setText("");

    panelResultados.removeAll();

    panelResultados.revalidate();

    panelResultados.repaint();

    filtroBox.setSelectedIndex(0);

}

// Inicializa los componentes de la interfaz

```

```

private void inicializarComponentes() {

    // Panel superior con título

    JPanel panelTitulo = new JPanel();

    panelTitulo.setLayout(new BoxLayout(panelTitulo, BoxLayout.Y_AXIS));

    panelTitulo.setBackground(new Color(25, 118, 210));

    panelTitulo.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0));


    JLabel lblTitulo = new JLabel("Sistema de Búsqueda de Libros");

    lblTitulo.setFont(new Font("Segoe UI", Font.BOLD, 28));

    lblTitulo.setForeground(Color.WHITE);

    lblTitulo.setAlignmentX(Component.CENTER_ALIGNMENT);


    JLabel lblSubtitulo = new JLabel("Biblioteca Virtual");

    lblSubtitulo.setFont(new Font("Segoe UI Light", Font.PLAIN, 16));

    lblSubtitulo.setForeground(new Color(224, 224, 224));

    lblSubtitulo.setAlignmentX(Component.CENTER_ALIGNMENT);


    panelTitulo.add(lblTitulo);

    panelTitulo.add(Box.createRigidArea(new Dimension(0, 5)));

    panelTitulo.add(lblSubtitulo);


    // Panel de búsqueda

    JPanel panelBusqueda = new JPanel(new GridBagLayout());

    panelBusqueda.setBackground(Color.WHITE);

    panelBusqueda.setBorder(BorderFactory.createCompoundBorder(

```

```

        BorderFactory.createLineBorder(new Color(200, 200, 200), 1),
        BorderFactory.createEmptyBorder(20, 20, 20, 20)
    ));

```

```

GridBagConstraints gbc = new GridBagConstraints();

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.insets = new Insets(5, 5, 5, 5);

```

```

// Campo de búsqueda con icono

```

```

JPanel searchPanel = new JPanel(new BorderLayout(5, 0));

searchPanel.setBackground(Color.WHITE);

txtBusqueda = new JTextField(30);

txtBusqueda.setFont(new Font("Segoe UI", Font.PLAIN, 14));

txtBusqueda.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(new Color(200, 200, 200)),
    BorderFactory.createEmptyBorder(8, 10, 8, 10)
));

txtBusqueda.addActionListener(e -> buscarLibros());

```

```

// ComboBox personalizado

```

```

String[] opciones = {"Todos", "Título", "Autor", "Precio"};

filtroBox = new JComboBox<>(opciones);

filtroBox.setFont(new Font("Segoe UI", Font.PLAIN, 14));

filtroBox.setPreferredSize(new Dimension(150, 38));

filtroBox.setBackground(Color.WHITE);

```



```

// Botones modernos

btnBuscar = new JButton("Buscar");

btnLimpiar = new JButton("Limpiar");


// Eventos

btnBuscar.addActionListener(e -> buscarLibros());

btnLimpiar.addActionListener(e -> limpiarResultados());


// Estilo botones

configurarBoton(btnBuscar, new Color(25, 118, 210));

configurarBoton(btnLimpiar, new Color(158, 158, 158));


// Layout

gbc.gridx = 0; gbc.gridy = 0;

panelBusqueda.add(new JLabel("Buscar:"), gbc);


gbc.gridx = 1; gbc.weightx = 1.0;

panelBusqueda.add(txtBusqueda, gbc);


gbc.gridx = 2; gbc.weightx = 0;

panelBusqueda.add(new JLabel("Filtrar por:"), gbc);


gbc.gridx = 3;

panelBusqueda.add(filtroBox, gbc);

```

```

gbc.gridx = 4;

panelBusqueda.add(btnBuscar, gbc);


gbc.gridx = 5;

panelBusqueda.add(btnLimpiar, gbc);


// Área de resultados

panelResultados = new JPanel();

panelResultados.setLayout(new GridLayout(0, 2, 10, 10)); // 2 columnas,
espaciado 10px

panelResultados.setBackground(new Color(240, 242, 245));


JScrollPane scrollResultados = new JScrollPane(panelResultados);
scrollResultados.setBorder(BorderFactory.createCompoundBorder(

    BorderFactory.createTitledBorder(null, "Resultados",

        TitledBorder.DEFAULT_JUSTIFICATION,

        TitledBorder.DEFAULT_POSITION,

        new Font("Segoe UI", Font.BOLD, 14)),

    BorderFactory.createEmptyBorder(10, 10, 10, 10)

));

scrollResultados.setPreferredSize(new Dimension(0, 400));


// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout(15, 15));

mainPanel.setBackground(new Color(240, 242, 245));

```

```

mainPanel.add(panelTitulo, BorderLayout.NORTH);

mainPanel.add(panelBusqueda, BorderLayout.CENTER);

mainPanel.add(scrollResultados, BorderLayout.SOUTH);


add(mainPanel);

}


// Añadir método auxiliar para configurar botones

private void configurarBoton(JButton btn, Color color) {

    btn.setFont(new Font("Segoe UI", Font.BOLD, 14));

    btn.setPreferredSize(new Dimension(120, 38));

    btn.setBackground(color);

    btn.setForeground(Color.WHITE);

    btn.setFocusPainted(false);

    btn.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));

    btn.setCursor(new Cursor(Cursor.HAND_CURSOR));


    btn.addMouseListener(new java.awt.event.MouseAdapter() {

        public void mouseEntered(java.awt.event.MouseEvent evt) {

            btn.setBackground(color.brighter());

        }

        public void mouseExited(java.awt.event.MouseEvent evt) {

            btn.setBackground(color);

        }

    });
}

```

```

}

// Realiza la búsqueda de libros

private void buscarLibros() {

    String termino = txtBusqueda.getText().trim();

    String filtro = (String) filtroBox.getSelectedItem();

    if (termino.isEmpty()) {

        JOptionPane.showMessageDialog(this,

            "Por favor ingrese un término de búsqueda",

            "Advertencia",

            JOptionPane.WARNING_MESSAGE);

        return;

    }

    try {

        List<String> resultados = servicioLibros.buscarLibros(termino);

        mostrarResultadosFiltrados(resultados, filtro);

    } catch (Exception e) {

        JOptionPane.showMessageDialog(this,

            "Error al buscar libros: " + e.getMessage(),

            "Error",

            JOptionPane.ERROR_MESSAGE);

    }

}

```

```

// Muestra los resultados filtrados en la interfaz

private void mostrarResultadosFiltrados(List<String> resultados, String filtro) {

    panelResultados.removeAll();

    if (resultados.isEmpty()) {

        JLabel noResultados = new JLabel("No se encontraron libros.");
        noResultados.setFont(new Font("Segoe UI", Font.PLAIN, 14));
        panelResultados.add(noResultados);
        panelResultados.revalidate();
        panelResultados.repaint();

        return;
    }

    for (String libro : resultados) {

        JPanel card = crearCard(libro, filtro);

        if (card != null) {

            panelResultados.add(card);

        }

    }

    panelResultados.revalidate();
    panelResultados.repaint();

}

```

```

// Crea una tarjeta para mostrar la información de un libro

private JPanel crearCard(String libro, String filtro) {

    JPanel card = new JPanel();

    card.setLayout(new BoxLayout(card, BoxLayout.Y_AXIS));

    card.setBackground(Color.WHITE);

    card.setBorder(BorderFactory.createCompoundBorder(

        BorderFactory.createLineBorder(new Color(200, 200, 200), 1),

        BorderFactory.createEmptyBorder(15, 15, 15, 15)

    ));

    // Cambiar el split para usar una expresión regular que mantenga el precio
    completo

    String[] partes = libro.split("(?<=\\D),|(?=\\D)");

    boolean mostrarCard = false;

    switch (filtro) {

        case "Todos":

            for (String parte : partes) {

                agregarCampoACard(card, parte.trim());

            }

            mostrarCard = true;

            break;

        case "Título":

            for (String parte : partes) {

                if (parte.trim().startsWith("Título:")) {

```

```

        agregarCampoACard(card, parte.trim());

        mostrarCard = true;

    }

}

break;

case "Autor":

    for (String parte : partes) {

        if (parte.trim().startsWith("Autor:")) {

            agregarCampoACard(card, parte.trim());

            mostrarCard = true;

        }

    }

    break;

case "Precio":

    for (String parte : partes) {

        if (parte.trim().startsWith("Precio:")) {

            agregarCampoACard(card, parte.trim());

            mostrarCard = true;

        }

    }

    break;

}

```

```

        return mostrarCard ? card : null;
    }

    // Añade un campo a la tarjeta
    private void agregarCampoACard(JPanel card, String texto) {

        JLabel label = new JLabel(texto);

        label.setFont(new Font("Segoe UI", Font.PLAIN, 14));

        label.setAlignmentX(Component.LEFT_ALIGNMENT);

        label.setBorder(BorderFactory.createEmptyBorder(2, 0, 2, 0));

        // Aplicar estilos según el tipo de información
        if (texto.startsWith("Título:")) {

            label.setFont(new Font("Segoe UI", Font.BOLD, 16));

            label.setForeground(new Color(25, 118, 210));

        } else if (texto.startsWith("Autor:")) {

            label.setForeground(new Color(0, 100, 0));

        } else if (texto.startsWith("Precio:")) {

            label.setForeground(new Color(178, 34, 34));

            label.setFont(new Font("Segoe UI", Font.BOLD, 14));

        }

        card.add(label);

        card.add(Box.createRigidArea(new Dimension(0, 5)));

    }

```



```

public static void main(String[] args) {

    SwingUtilities.invokeLater(() -> {

        InterfazCliente cliente = new InterfazCliente();

        cliente.setLocationRelativeTo(null);

        cliente.setVisible(true);

    });

}
}

```

## | | └─ **InterfazServidor.java**

```

package interfaz;

import java.awt.*;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.*;

import servidor.IServicioLibros;

import servidor.ServidorLibros;

public class InterfazServidor extends JFrame {

    private JButton btnIniciar;

    private JButton btnDetener;

    private JTextArea areaLogs;

```

```

private JLabel lblEstado;

private JLabel lblConexion;

private Registry registry;

private IServicioLibros stub;

private ServidorLibros servicioLibros;

private boolean servidorIniciado = false;


public InterfazServidor() {

    // Configura la ventana principal

    configurarVentana();

    // Inicializa los componentes de la interfaz

    inicializarComponentes();

    // Configura los eventos de los botones

    configurarEventos();

    // Verifica el estado del servidor al iniciar

    verificarEstadoServidor();

}


// Añadir nuevo método para verificación

private void verificarEstadoServidor() {

    try {

        // Intenta conectarse al registro RMI en el puerto 1099

        Registry testRegistry = LocateRegistry.getRegistry(1099);

        try {

            // Intenta buscar el servicio

```

```

testRegistry.lookup("ServicioLibros");

// Si no lanza excepción, el servidor está activo

servidorIniciado = true;

btnIniciar.setEnabled(false);

btnDetener.setEnabled(true);

lblEstado.setText("Estado: Ejecutando");

lblEstado.setForeground(new Color(46, 139, 87));

lblConexion.setText("Base de datos: Conectada");

lblConexion.setForeground(new Color(46, 139, 87));

agregarLog("Servidor detectado en ejecución");

} catch (Exception e) {

    // El registro existe pero el servicio no está disponible

    agregarLog("Puerto 1099 en uso pero el servicio no está disponible");

}

} catch (Exception e) {

    // El registro no existe, el servidor está completamente detenido

    servidorIniciado = false;

    agregarLog("Servidor no detectado");

}

}

// Modificar el método iniciarServidor()

private void iniciarServidor() throws Exception {

    if (servidorIniciado) {

        agregarLog("Aviso: El servidor ya está en ejecución");

```

```

JOptionPane.showMessageDialog(this,

    "El servidor ya está en ejecución",

    "Aviso",

    JOptionPane.INFORMATION_MESSAGE);

return;

}

try {

    // Primero intentamos limpiar cualquier instancia anterior

    try {

        Registry registroExistente = LocateRegistry.getRegistry(1099);

        try {

            registroExistente.unbind("ServicioLibros");

        } catch (Exception e) {

            // Ignoramos si el servicio no existe

        }

    } catch (Exception e) {

        // Ignoramos si no hay registro existente

    }

    // Esperamos un momento para asegurar que los recursos se liberan

    Thread.sleep(100);

    // Creamos una nueva instancia del servicio

    servicioLibros = new ServidorLibros();

```

```

// Intentamos crear un nuevo registro

try {

    registry = LocateRegistry.createRegistry(1099);

} catch (Exception e) {

    // Si falla, obtenemos el registro existente

    registry = LocateRegistry.getRegistry(1099);

}


// Exportamos el objeto y lo vinculamos al registro

stub = (IServicioLibros) UnicastRemoteObject.exportObject(servicioLibros, 0);

registry.rebind("ServicioLibros", stub);


servidorIniciado = true;

lblConexion.setText("Base de datos: Conectada");

lblConexion.setForeground(new Color(46, 139, 87));

agregarLog("Servidor iniciado correctamente");


} catch (Exception e) {

    // Si algo falla, intentamos limpiar todo

    if (servicioLibros != null) {

        try {

            UnicastRemoteObject.unexportObject(servicioLibros, true);

        } catch (Exception ex) {

            // Ignoramos errores de limpieza


```

```

        }

    }

    throw new Exception(e.getMessage());

}

}

// Modificar el método detenerServidor()

private void detenerServidor() throws Exception {

    try {

        if (!servidorIniciado) {

            throw new Exception("El servidor no está en ejecución");

        }

        if (registry != null) {

            // Desvincula el objeto del servicio del registro

            registry.unbind("ServicioLibros");

            if (stub != null) {

                // Desexporta el objeto del servicio

                UnicastRemoteObject.unexportObject(servicioLibros, true);

            }

            // Desexporta el registro

            UnicastRemoteObject.unexportObject(registry, true);

            registry = null;

            stub = null;

            servicioLibros = null;

            servidorIniciado = false;

```

```

        lblConexion.setText("Base de datos: No conectada");

        lblConexion.setForeground(Color.RED);

    }

} catch (Exception e) {

    throw new Exception("Error al detener el servidor: " + e.getMessage());

}

}

// Modificar la parte relevante de configurarEventos()

private void configurarEventos() {

    btnIniciar.addActionListener(e -> {

        try {

            iniciarServidor();

            // Solo actualizar la UI si el servidor se inició correctamente

            if (servidorIniciado) {

                btnIniciar.setEnabled(false);

                btnDetener.setEnabled(true);

                lblEstado.setText("Estado: Ejecutando");

                lblEstado.setForeground(new Color(46, 139, 87));

                agregarLog("Servidor iniciado correctamente");

            }

        } catch (Exception ex) {

            agregarLog("Error al iniciar el servidor: " + ex.getMessage());

            JOptionPane.showMessageDialog(this,

                "Error al iniciar el servidor: " + ex.getMessage(),

```

```

        "Error",

        JOptionPane.ERROR_MESSAGE);

    }

});

btnDetener.addActionListener(e -> {

    try {

        detenerServidor();

        btnIniciar.setEnabled(true);

        btnDetener.setEnabled(false);

        lblEstado.setText("Estado: Detenido");

        lblEstado.setForeground(Color.RED);

        agregarLog("Servidor detenido correctamente");

    } catch (Exception ex) {

        agregarLog("Error al detener el servidor: " + ex.getMessage());

        if (!ex.getMessage().contains("no está en ejecución")) {

            JOptionPane.showMessageDialog(this,

                "Error al detener el servidor: " + ex.getMessage(),

                "Error",

                JOptionPane.ERROR_MESSAGE);

        }

    }

});

}

```



```

private void agregarLog(String mensaje) {

    String timestamp = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date());

    areaLogs.append(timestamp + " - " + mensaje + "\n");

    // Auto-scroll al final

    areaLogs.setCaretPosition(areaLogs.getDocument().getLength());

}

private void configurarVentana() {

    setTitle("Panel de Control - Servidor de Biblioteca");

    setSize(800, 600);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setLayout(new BorderLayout(10, 10));

    getContentPane().setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

}

private void inicializarComponentes() {

    // Panel de título

    JPanel panelTitulo = new JPanel(new FlowLayout(FlowLayout.CENTER));

    JLabel titulo = new JLabel("Control del Servidor de Biblioteca");

    titulo.setFont(new Font("Arial", Font.BOLD, 24));

    panelTitulo.add(titulo);

    // Panel de control

    JPanel panelControl = new JPanel(new GridBagLayout());

    panelControl.setBorder(BorderFactory.createTitledBorder("Panel de Control"));

```

```

GridBagConstraints gbc = new GridBagConstraints();

// Botones

btnIniciar = new JButton("Iniciar Servidor");

btnDetener = new JButton("Detener Servidor");

btnIniciar.setPreferredSize(new Dimension(150, 40));

btnDetener.setPreferredSize(new Dimension(150, 40));

btnDetener.setEnabled(false);

// Estilo botones

btnIniciar.setBackground(new Color(46, 139, 87));

btnIniciar.setForeground(Color.WHITE);

btnDetener.setBackground(new Color(178, 34, 34));

btnDetener.setForeground(Color.WHITE);

// Estado y conexión

lblEstado = new JLabel("Estado: Detenido");

lblEstado.setFont(new Font("Arial", Font.BOLD, 14));

lblConexion = new JLabel("Base de datos: No conectada");

lblConexion.setFont(new Font("Arial", Font.PLAIN, 14));

// Configurar layout

gbc.gridx = 0; gbc.gridy = 0;

gbc.insets = new Insets(5,5,5,5);

panelControl.add(btnIniciar, gbc);

```

```

gbc.gridx = 1;

panelControl.add(btnDetener, gbc);


gbc.gridx = 0; gbc.gridy = 1;

gbc.gridwidth = 2;

panelControl.add(lblEstado, gbc);


gbc.gridy = 2;

panelControl.add(lblConexion, gbc);


// Área de logs

areaLogs = new JTextArea();

areaLogs.setEditable(false);

areaLogs.setFont(new Font("Monospaced", Font.PLAIN, 12));

JScrollPane scrollLogs = new JScrollPane(areaLogs);

scrollLogs.setBorder(BorderFactory.createTitledBorder("Registro de Eventos"));


// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout(10, 10));

mainPanel.add(panelTitulo, BorderLayout.NORTH);

mainPanel.add(panelControl, BorderLayout.CENTER);

mainPanel.add(scrollLogs, BorderLayout.SOUTH);


add(mainPanel);

```

```

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            InterfazServidor servidor = new InterfazServidor();

            servidor.setLocationRelativeTo(null);

            servidor.setVisible(true);

        });

    }
}

|  |—— servidor/
|  |  |—— IServicioLibros.java

package servidor;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.List;

public interface IServicioLibros extends Remote {

    // Método remoto para buscar libros por una clave

    List<String> buscarLibros(String clave) throws RemoteException;

}

|  |  |—— ServidorLibros.java

package servidor;

```

```

import datos.LibroDAO;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.List;


public class ServidorLibros implements IServicioLibros {

    private LibroDAO libroDAO;

    private IServicioLibros stub;


    // Constructor modificado que no extiende UnicastRemoteObject

    public ServidorLibros() {

        libroDAO = new LibroDAO();

    }


    // Método para exportar el objeto

    public IServicioLibros exportar() throws RemoteException {

        if (stub == null) {

            stub = (IServicioLibros) UnicastRemoteObject.exportObject(this, 0);

        }

        return stub;

    }


    // Método para desexportar el objeto

    public void desexportar() {

        try {

```

```

        if (stub != null) {

            UnicastRemoteObject.unexportObject(this, true);

            stub = null;

        }

    } catch (Exception e) {

        // Ignorar errores de desexportación

    }

}

// Implementación del método remoto para buscar libros

@Override

public List<String> buscarLibros(String clave) throws RemoteException {

    return libroDAO.buscarLibros(clave);

}

```

```
package servidor;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class ServidorManager {
```

```

private static final int PUERTO = 1099;

private Registry registry;

private ServidorLibros servidor;

// Verifica si el puerto está bloqueado

private boolean puertoBloqueado() {

    try {

        // Ejecuta un comando para verificar si el puerto está en uso

        ProcessBuilder pb = new ProcessBuilder(

            "cmd", "/c", "netstat", "-ano", "|", "findstr", String.valueOf(PUERTO));

        Process process = pb.start();

        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));

        String line = reader.readLine();

        // Si se encuentra una línea, el puerto está en uso

        return line != null && !line.isEmpty();

    } catch (Exception e) {

        return false;

    }

}

// Libera el puerto especificado matando el proceso que lo está usando

private void liberarPuerto(String pid) {

    try {

```

```

if (pid != null && !pid.isEmpty()) {

    // Ejecuta un comando para matar el proceso que está usando el puerto

    ProcessBuilder pb = new ProcessBuilder(

        "taskkill", "/F", "/PID", pid);

    pb.start();

}

} catch (Exception e) {

    System.err.println("Error al liberar el puerto: " + e.getMessage());

}

}

// Inicia el servidor RMI

public void iniciarServidor() throws Exception {

    try {

        // Primero intentamos limpiar cualquier registro existente

        try {

            Registry registroExistente = LocateRegistry.getRegistry(PUERTO);

            try {

                registroExistente.unbind("ServicioLibros");

            } catch (Exception e) {

                // Ignoramos si el servicio no existe

            }

        } catch (Exception e) {

            // Ignoramos si no hay registro existente

        }

    }

}

```



```

// Esperamos un momento para asegurar que los recursos se liberan
Thread.sleep(100);

// Creamos una nueva instancia del servidor
servidor = new ServidorLibros();

// Intentamos crear un nuevo registro
try {
    registry = LocateRegistry.createRegistry(PUERTO);
} catch (Exception e) {
    // Si falla, obtenemos el registro existente
    registry = LocateRegistry.getRegistry(PUERTO);
}

// Exportamos el objeto y lo vinculamos al registro
IServicioLibros stub = (IServicioLibros)
UnicastRemoteObject.exportObject(servidor, 0);

registry.rebind("ServicioLibros", stub);

} catch (Exception e) {
    // Si algo falla, intentamos limpiar todo
    if (servidor != null) {
        try {
            UnicastRemoteObject.unexportObject(servidor, true);
        } catch (Exception ex) {

```

```

        // Ignoramos errores de limpieza
    }

}

throw new Exception("Error al iniciar el servidor: " + e.getMessage());

}

}

// Detiene el servidor RMI

public void detenerServidor() throws Exception {

    try {

        if (registry != null) {

            try {

                registry.unbind("ServicioLibros");

            } catch (Exception e) {

                // Ignoramos si el servicio ya no existe

            }

        }

        if (servidor != null) {

            UnicastRemoteObject.unexportObject(servidor, true);

        }

        UnicastRemoteObject.unexportObject(registry, true);

        registry = null;

        servidor = null;

    }

    } catch (Exception e) {

```

```
        throw new Exception("Error al detener el servidor: " + e.getMessage());
    }
}
}
```