Proyecto Final Desarrollo de Software

Carlos A. Camacho Castaño – 2160331

Kevin Marín – 2160364

Kevin S. Ramírez - 2259371

Harrison I. Valencia – 2159979

Universidad del Valle – Sede Tuluá

Escuela de Ingeniería de Sistemas y Computación

Ing. Juan Pinillos

Tuluá, Valle del Cauca

Junio de 2024

Login (Interfaz de usuario):

Menú para ingresar usuario.

- Método para manejar el evento cuando se hace clic en el botón de ingresar. Valida el usuario y la contraseña ingresados, y muestra la ventana correspondiente según el tipo de usuario.

```
private void BtnIngresarMouseClicked(java.awt.event.MouseEvent evt) {
    if (campoUsuario.getText().isEmpty() || campoContra.getPassword().length == 0) {
        JOptionPane.showMessageDialog(null, "Por favor, completa todos los campos.");
        campoUsuario.setText("");
        campoContra.setText("");
        campoUsuario.requestFocus();
        return; // Salir del método si algún campo está vacío
    }

    String nombreIngresado = campoUsuario.getText();
    String contraseñaIngresada = new String(campoContra.getPassword());
    String tipoUsuario = campoTipo.getSelectedItem().toString();

    if (tipoUsuario.equals("Cliente")) {
        Usuario usuario = Usuario.validarUsuario(nombreIngresado, contraseñaIngresada);
        if (usuario != null) {
            JOptionPane.showMessageDialog(null, "¡Bienvenido Cliente!");
            Home firstScreen = new Home(usuario);
            firstScreen.setVisible(true);
            this.dispose(); // Cierra la ventana actual de login
        } else {
            JOptionPane.showMessageDialog(null, "Usuario o contraseña incorrectos.");
            campoContra.setText("");
            campoUsuario.requestFocus();
        }
    } else if (tipoUsuario.equals("Admin")) {
        AdminC admin = AdminC.validarAdmin(nombreIngresado, contraseñaIngresada);
        if (admin != null) {
            JOptionPane.showMessageDialog(null, "¡Bienvenido Admin!");
            Admin thirdScreen = new Admin();
            thirdScreen.setVisible(true);
            this.dispose(); // Cierra la ventana actual de login
        } else {
            JOptionPane.showMessageDialog(null, "Usuario o contraseña incorrectos.");
            campoContra.setText("");
            campoUsuario.requestFocus();
        }
    }
}
```

Registrar (Interfaz de usuario):

Menú para el apartado de registro

- Implementación del metodo para el registro de usuarios en el sistema.

```
private void BtnRegistrarMouseClicked(java.awt.event.MouseEvent evt) {
    String dni = TfdDni.getText().trim();
    String nombre = Tfdusuario.getText().trim();
    String correo = Tfdcorreo.getText().trim();
    String contraseña = new String(Tfdcontraseña.getPassword());
    String confirmarContraseña = new String(TfdConfirContraseña.getPassword());

    // Validación de campos vacíos
    if (dni.isEmpty() || nombre.isEmpty() || correo.isEmpty() || contraseña.isEmpty() || confirmarContraseña.isEmpty()) {
        JOptionPane.showMessageDialog(null, "Por favor, completa todos los campos.");
        TfdDni.setText("");
        Tfdusuario.setText("");
        Tfdcorreo.setText("");
        Tfdcontraseña.setText("");
        TfdConfirContraseña.setText("");
        TfdDni.requestFocus();
        return;
    }

    // Validación de coincidencia de contraseñas
    if (!contraseña.equals(confirmarContraseña)) {
        JOptionPane.showMessageDialog(null, "¡Las contraseñas no coinciden!");
        Tfdcontraseña.setText("");
        TfdConfirContraseña.setText("");
        Tfdcontraseña.requestFocus();
        return;
    }

    try {
        // Verificar si el usuario ya está registrado por DNI, nombre de usuario o correo electrónico
        if (usuarioExiste("dni", dni)) {
            JOptionPane.showMessageDialog(null, "¡El DNI ya está registrado!");
            TfdDni.setText("");
            TfdDni.requestFocus();
            return;
        }

        if (usuarioExiste("nombre", nombre)) {
            JOptionPane.showMessageDialog(null, "¡El nombre de usuario ya está registrado!");
            Tfdusuario.setText("");
            Tfdusuario.requestFocus();
            return;
        }

        if (usuarioExiste("correo", correo)) {
            JOptionPane.showMessageDialog(null, "¡El correo electrónico ya está registrado!");
            Tfdcorreo.setText("");
            Tfdcorreo.requestFocus();
            return;
        }

        // Registrar el usuario si no existe y enviar correo de confirmación
        if (registrarUsuario(dni, nombre, correo, contraseña)) {
            enviarCorreoDeConfirmacion(correo);
            JOptionPane.showMessageDialog(null, "¡Te has registrado exitosamente! Se ha enviado un correo de bienvenida.");
            limpiarCampos();
        } else {
            JOptionPane.showMessageDialog(null, "¡Hubo un problema al registrar el usuario!");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "¡Hubo un error al acceder a la base de datos!");
    }

}
```

Home (Interfaz de usuario):
Menu principal de los clientes.

- Método “LlenarTabla”
  Llena la tabla de libros con la lista proporcionada.
  @param libros Lista de libros que se mostrarán en la tabla

```
private void llenarTabla(List<Libro> libros) {
    mt = (DefaultTableModel) jTable1.getModel();
    mt.setRowCount(0);

    for (Libro libro : libros) {
        mt.addRow(new Object[]{
            libro.getCodigo(),
            libro.getTitulo(),
            libro.getEstado(),
            libro.getCategoria(),
            libro.getAutor(),
            libro.getAnoLanzamiento()
        });
    }
}
```

- Método “llenarCategorias”
  Llena el combo box de categorías con las categorías disponibles.

```
private void llenarCategorias() {
    Set<String> categorias = obtenerCategorias();
    DefaultComboBoxModel<String> model = new DefaultComboBoxModel<>();
    model.addElement("Todas");
    for (String categoria : categorias) {
        model.addElement(categoria);
    }
    JcbCategoria.setModel(model);
}
```

- Método “obtenerCategorias”
  Obtiene todas las categorías distintas de los libros almacenados en la base de datos.
  @return Conjunto de cadenas que representan las categorías disponibles

```
private Set<String> obtenerCategorias() {
    Set<String> categorias = new HashSet<>();
    String query = "SELECT DISTINCT categoria FROM libros";

    try (Statement stmt = connection.createStatement();
         ResultSet rs = stmt.executeQuery(query)) {
        while (rs.next()) {
            categorias.add(rs.getString("categoria"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return categorias;
}
```

- Método “mostrarLibrosSolicitados”
  Muestra los libros solicitados por el usuario logueado en la tabla

```
private void mostrarLibrosSolicitados() {
    modeloSolicitudes = new DefaultTableModel(
        new Object[][] {},
        new String[] { "Código", "Título", "Fecha Préstamo", "Fecha Devolución", "Estado" }
    ) {
        boolean[] canEdit = new boolean[] { false, false, false, false, true };

        @Override
        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit[columnIndex];
        }
    };
    JtaMostrarSolicitudes.setModel(modeloSolicitudes);

    // Obtener y mostrar los préstamos del usuario logueado
    List<Prestamo> prestamos = obtenerPrestamos(usuarioLogueado.getDni());
    for (Prestamo prestamo : prestamos) {
        String codigoLibro = prestamo.getCodigoLibro();
        String estadoLibro = prestamo.getEstado();
        String tituloLibro = obtenerTituloLibroPorCodigo(codigoLibro);
        String fechaPrestamo = prestamo.getFechaPrestamo() != null ? prestamo.getFechaPrestamo().toString() : "N/A";
        String fechaDevolucion = prestamo.getFechaDevolucion() != null ? prestamo.getFechaDevolucion().toString() : "N/A";
        modeloSolicitudes.addRow(new Object[] { codigoLibro, tituloLibro, fechaPrestamo, fechaDevolucion, estadoLibro });
    }
}
```

- Metodo “solicitarPrestamo”
  Realiza la solicitud de un prestamo realizado por el usuario

```
private void solicitarPrestamo() {
    if (usuarioLogueado == null) {
        JOptionPane.showMessageDialog(this, "Debe estar logueado para solicitar un préstamo", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    int selectedRow = jTable1.getSelectedRow();
    if (selectedRow == -1) {
        JOptionPane.showMessageDialog(this, "Debe seleccionar un libro para solicitar un préstamo", "Error", JOptionPane.ERROR_MESSA
        return;
    }

    // Obtener información del libro seleccionado
    String codigoLibro = (String) jTable1.getValueAt(selectedRow, 0);
    String estadoLibro = (String) jTable1.getValueAt(selectedRow, 2);

    // Verificar el estado del libro seleccionado
    if ("Prestado".equalsIgnoreCase(estadoLibro) || "Pendiente".equalsIgnoreCase(estadoLibro)) {
        JOptionPane.showMessageDialog(this, "El libro seleccionado ya está prestado o en espera de aprobación", "Error", JOptionPane
        return;
    }

    // Consultas preparadas para insertar el préstamo y actualizar el estado del libro
    String insertPrestamo = "INSERT INTO prestamos (dni_usuario, codigo_libro, fecha_prestamo, fecha_devolucion, estado) VALUES (?,
    String updateLibro = "UPDATE libros SET estado = ? WHERE codigo = ?";

    try (PreparedStatement psPrestamo = connection.prepareStatement(insertPrestamo);
         PreparedStatement psLibro = connection.prepareStatement(updateLibro)) {

        // Iniciar transacción
        connection.setAutoCommit(false);

        // Insertar préstamo en la tabla prestamos
        psPrestamo.setString(1, usuarioLogueado.getDni());
        psPrestamo.setString(2, codigoLibro);
        psPrestamo.setDate(3, java.sql.Date.valueOf(LocalDate.now()));
        psPrestamo.setDate(4, null);
        psPrestamo.setString(5, "Pendiente");
        psPrestamo.executeUpdate();

        // Actualizar estado del libro en la tabla libros
        psLibro.setString(1, "Pendiente");
        psLibro.setString(2, codigoLibro);
        psLibro.executeUpdate();

        // Confirmar la transacción
        connection.commit();

        // Mostrar mensaje de éxito
        JOptionPane.showMessageDialog(this, "Solicitud de préstamo realizada con éxito", "Éxito", JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        // En caso de error, realizar rollback y manejar excepción
        try {
            connection.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        // Restaurar el modo de autocommit al finalizar
        try {
            connection.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Actualizar la tabla de libros y mostrar los préstamos solicitados
    llenarTabla(obtenerLibros());
    mostrarLibrosSolicitados();
}
```

- Método “ObtenerTituloLibroPorCodigo”
  Obtiene el título de un libro específico a partir de su código.
  @param codigoLibro Código del libro del cual obtener el título.
  @return Título del libro encontrado o "Título no encontrado" si no se encuentra

```
private String obtenerTituloLibroPorCodigo(String codigoLibro) {
    for (Libro libro : obtenerLibros()) {
        if (libro.getCodigo().equals(codigoLibro)) {
            return libro.getTitulo();
        }
    }
    return "Título no encontrado";
}
```

- Método “ObtenerLibros”
  Obtiene todos los libros disponibles en la base de datos.
  @return Lista de libros disponibles.

```
private List<Libro> obtenerLibros() {
    List<Libro> libros = new ArrayList<>();
    String query = "SELECT * FROM libros";

    try (PreparedStatement ps = connection.prepareStatement(query);
         ResultSet rs = ps.executeQuery()) {
        // Crear un objeto Libro a partir de los datos de la base de datos
        while (rs.next()) {
            Libro libro = new Libro(
                rs.getString("codigo"),
                rs.getString("titulo"),
                rs.getString("estado"),
                rs.getString("categoria"),
                rs.getString("autor"),
                rs.getInt("ano_lanzamiento")
            );
            libros.add(libro);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return libros;
}
```

- Método “devolverLibro”
  Realiza la devolución de un libro previamente prestado por el usuario logueado.

```
private void devolverLibro() {
    if (usuarioLogueado == null) {
        JOptionPane.showMessageDialog(this, "Debe estar logueado para devolver un libro", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    int selectedRow = JtaMostrarSolicitudes.getSelectedRow();
    if (selectedRow == -1) {
        JOptionPane.showMessageDialog(this, "Debe seleccionar un libro para devolver", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    String codigoLibro = (String) JtaMostrarSolicitudes.getValueAt(selectedRow, 0);
    String estadoPrestamo = (String) JtaMostrarSolicitudes.getValueAt(selectedRow, 4);

    // Verificar que el libro esté en estado "Prestado"
    if (!"Prestado".equalsIgnoreCase(estadoPrestamo)) {
        JOptionPane.showMessageDialog(this, "El libro seleccionado no está en estado 'Prestado'", "Error", JOptionPane.ERROR_MESSAGE
        return;
    }

    // Consultas preparadas para eliminar el préstamo y actualizar el estado del libro
    String deletePrestamo = "DELETE FROM prestamos WHERE dni_usuario = ? AND codigo_libro = ?";
    String updateLibro = "UPDATE libros SET estado = ? WHERE codigo = ?";

    try (PreparedStatement psPrestamo = connection.prepareStatement(deletePrestamo);
         PreparedStatement psLibro = connection.prepareStatement(updateLibro)) {

        // Iniciar transacción
        connection.setAutoCommit(false);

        // Eliminar el préstamo de la tabla prestamos
        psPrestamo.setString(1, usuarioLogueado.getDni());
        psPrestamo.setString(2, codigoLibro);
        psPrestamo.executeUpdate();

        // Actualizar estado del libro en la tabla libros a "Disponible"
        psLibro.setString(1, "Disponible");
        psLibro.setString(2, codigoLibro);
        psLibro.executeUpdate();

        // Confirmar la transacción
        connection.commit();

        // Mostrar mensaje de éxito
        JOptionPane.showMessageDialog(this, "Libro devuelto con éxito", "Éxito", JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        // En caso de error, realizar rollback y manejar excepción
        try {
            connection.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        // Restaurar el modo de autocommit al finalizar
        try {
            connection.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Actualizar la tabla de libros y mostrar los libros solicitados
    llenarTabla(obtenerLibros());
    mostrarLibrosSolicitados();
}
```

Admin (Interfaz de usuario):

- Método al darle click al boton “Agregar”
  Agrega un libro nuevo al inventario.

```
private void JbAgregarMouseClicked(java.awt.event.MouseEvent evt) {
    // Obtiene los valores de los campos de texto
    String titulo = JtfTitulo.getText();
    String estado = JtfEstado.getText();
    String categoria = JtfCategoria.getText();
    String autor = JtfAutor.getText();
    String anioText = JtfAñoLanza.getText();
    String codigo = JtfCodigo.getText();

    // Verifica que todos los campos estén completos
    if (codigo.isEmpty() || titulo.isEmpty() || estado.isEmpty() || categoria.isEmpty() || autor.isEmpty() || anioText.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Por favor, complete todos los campos antes de agregar un libro.");
        return;
    }

    // Verifica si el código es único
    try (PreparedStatement ps = connection.prepareStatement("SELECT COUNT(*) FROM libros WHERE codigo = ?")) {
        ps.setString(1, codigo);
        ResultSet rs = ps.executeQuery();
        if (rs.next() && rs.getInt(1) > 0) {
            JOptionPane.showMessageDialog(this, "El código del libro ya existe en el catálogo. Por favor, ingrese un código diferente.");
            JtfCodigo.setText("");
            JtfCodigo.requestFocus();
            return;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    // Convierte el año a un entero
    int anio;
    try {
        anio = Integer.parseInt(anioText);
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "El año ingres// Limpia los campos de texto después de agregar el libroado no es válido. Por favor, 
        JtfAñoLanza.setText("");
        JtfAñoLanza.requestFocus();
        return;
    }

    // Inserta el nuevo libro en la base de datos
    try (PreparedStatement ps = connection.prepareStatement(
        "INSERT INTO libros (codigo, titulo, estado, categoria, autor, ano_lanzamiento) VALUES (?, ?, ?, ?, ?, ?)")) {
        ps.setString(1, codigo);
        ps.setString(2, titulo);
        ps.setString(3, estado);
        ps.setString(4, categoria);
        ps.setString(5, autor);
        ps.setInt(6, anio);
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    // Limpia los campos de texto después de agregar el libro
    JtfTitulo.setText("");
    JtfEstado.setText("");
    JtfCategoria.setText("");
    JtfAutor.setText("");
    JtfAñoLanza.setText("");
    JtfCodigo.setText("");

    JOptionPane.showMessageDialog(this, "El libro ha sido agregado exitosamente.");
    cargarTabla();
}
```

- Método al darle click al boton “Actualizar”
  Modifica/Actualiza los datos del libro seleccionado

```
private void JbActualizarMouseClicked(java.awt.event.MouseEvent evt) {
    // Obtiene la fila seleccionada de la tabla
    int selectedRow = JtaLibros.getSelectedRow();
    if (selectedRow != -1) {
        // Verifica y obtiene los valores de los campos de texto
        String codigo = JtfCodigo.getText().equals("Codigo") ? generarCodigoUnico() : JtfCodigo.getText();
        String titulo = JtfTitulo.getText().isEmpty() ? null : JtfTitulo.getText();
        String estado = JtfEstado.getText().isEmpty() ? null : JtfEstado.getText();
        String categoria = JtfCategoria.getText().isEmpty() ? null : JtfCategoria.getText();
        String autor = JtfAutor.getText().isEmpty() ? null : JtfAutor.getText();
        String anioText = JtfAñoLanza.getText().isEmpty() ? null : JtfAñoLanza.getText();

        // Verifica si alguno de los campos tiene el valor predeterminado
        if (titulo.equals("Titulo")||estado.equals("Estado")||categoria.equals("Categoria")||autor.equals("Autor")) {
            JOptionPane.showMessageDialog(this, "Por favor llene los capos.");
            // Limpia y enfoca los campos incorrectos
            if(titulo.equals("Titulo")){
                JtfTitulo.setText("");
                JtfTitulo.requestFocus();
                return;
            }
            if(estado.equals("Estado")){
                JtfEstado.setText("");
                JtfEstado.requestFocus();
                return;
            }
            if(categoria.equals("Categoria")){
                JtfCategoria.setText("");
                JtfCategoria.requestFocus();
                return;
            }
            if(autor.equals("Autor")){
                JtfAutor.setText("");
                JtfAutor.requestFocus();
                return;
            }
        }

        // Verifica si el código es único
        if (codigo != null) {
            try (PreparedStatement ps = connection.prepareStatement("SELECT COUNT(*) FROM libros WHERE codigo = ?")) {
                ps.setString(1, codigo);
                ResultSet rs = ps.executeQuery();
                if (rs.next() && rs.getInt(1) > 0) {
                    JOptionPane.showMessageDialog(this, "El código del libro ya existe en el catálogo. Por favor, ingrese un código diferente.");
                    JtfCodigo.setText("");
                    JtfCodigo.requestFocus();
                    return;
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
        // Convierte el año a un entero, si es posible
        Integer anio = null;
        if (anioText != null) {
            try {
                anio = Integer.parseInt(anioText);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(this, "Por favor, Ingrese el año de lanzamiento.");
                JtfAñoLanza.setText("");
                JtfAñoLanza.requestFocus();
                return;
            }
        }

        // Obtiene el código del libro seleccionado
        String codigoSeleccionado = (String) mt.getValueAt(selectedRow, 0);

        // Actualiza la información del libro en la base de datos
        try (PreparedStatement ps = connection.prepareStatement(
            "UPDATE libros SET codigo = ?, titulo = ?, estado = ?, categoria = ?, autor = ?, ano_lanzamiento = ? WHERE codigo = ?")) {
            ps.setString(1, codigo);
            ps.setString(2, titulo);
            ps.setString(3, estado);
            ps.setString(4, categoria);
            ps.setString(5, autor);
            if (anio != null) {
                ps.setInt(6, anio);
            } else {
                ps.setNull(6, Types.INTEGER);
            }
             ps.setString(7, codigoSeleccionado);
             ps.executeUpdate();
         } catch (SQLException e) {
             e.printStackTrace();
         }

         // Limpia los campos de texto después de la actualización
         JtfTitulo.setText("");
         JtfEstado.setText("");
         JtfCategoria.setText("");
         JtfAutor.setText("");
         JtfAñoLanza.setText("");
         JtfCodigo.setText("");

         JOptionPane.showMessageDialog(this, "El libro ha sido actualizado exitosamente.");
         cargarTabla();
     } else {
         JOptionPane.showMessageDialog(this, "Por favor, seleccione un libro para actualizar.");
     }
 }
```

- Método al presionar el boton “Eliminar”
  Elimina los datos de un libro

```
private void JbEliminarMouseClicked(java.awt.event.MouseEvent evt) {
    // Obtiene la fila seleccionada de la tabla
    int selectedRow = JtaLibros.getSelectedRow();
    if (selectedRow != -1) {
        String codigo = (String) mt.getValueAt(selectedRow, 0);

        // Elimina el libro de la base de datos
        try (PreparedStatement ps = connection.prepareStatement("DELETE FROM libros WHERE codigo = ?")) {
            ps.setString(1, codigo);
            ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        JOptionPane.showMessageDialog(this, "El libro ha sido eliminado exitosamente.");
        cargarTabla();
    } else {
        JOptionPane.showMessageDialog(this, "Por favor, seleccione un libro para eliminar.");
    }
}
```

- Metodo al presionar el boton “Prestar”
  Registra un nuevo prestamo en el sistema

```java
private void JbPrestarMouseClicked(java.awt.event.MouseEvent evt) {
    // Verifica si se ha seleccionado una solicitud de préstamo
    if (JtaMostrarSolicitudes.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(this, "Por favor, seleccione un usuario con su libro que desea Prestar.");
        return;
    }

    int selectedRow = JtaMostrarSolicitudes.getSelectedRow();
    String codigoLibro = (String) JtaMostrarSolicitudes.getValueAt(selectedRow, 1);

    // Verifica si se ha ingresado una fecha de devolución
    if (JtfFechaDevolucion.getText().isEmpty() || JtfFechaDevolucion.getText().equals("YYYY-MM-DD")) {
        JOptionPane.showMessageDialog(this, "Por favor, ingrese una fecha de devolución.");
        JtfFechaDevolucion.requestFocus();
        return;
    }

    // // Obtiene la fecha de devolución ingresada
    LocalDate fechaDevolucion = LocalDate.parse(JtfFechaDevolucion.getText());

    // Verifica y actualiza el estado del libro en la base de datos
    try {
        connection.setAutoCommit(false);

        // Verifica si el libro ya está prestado
        try (PreparedStatement ps = connection.prepareStatement("SELECT estado FROM prestamos WHERE id = ? AND estado = 'Prestado'")) {
            ps.setInt(1, Integer.parseInt(codigoLibro));
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                JOptionPane.showMessageDialog(this, "El libro no se encuentra disponible para préstamo.");
                JtfFechaDevolucion.setText("");
                connection.rollback();
                return;
            }
        }

        // Actualiza el estado del libro en la tabla prestamos
        try (PreparedStatement ps = connection.prepareStatement("UPDATE prestamos SET estado = 'Prestado', fecha_prestamo = ?, fecha_devolucion = ? WHERE codigo_libro = ?")) {
            ps.setDate(1, Date.valueOf(LocalDate.now()));
            ps.setDate(2, Date.valueOf(fechaDevolucion));
            ps.setString(3, codigoLibro);
            ps.executeUpdate();
        }

        // Actualiza el estado del libro en la tabla libros
        try (PreparedStatement ps = connection.prepareStatement("UPDATE libros SET estado = 'Prestado' WHERE codigo = ?")) {
            ps.setString(1, codigoLibro);
            ps.executeUpdate();
        }

        connection.commit();
    } catch (SQLException e) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        try {
            connection.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    JtfFechaDevolucion.setText("");
    JtfFechaDevolucion.requestFocus();
    cargarTabla();
    mostrarLibrosSolicitados();
}
```

- Método al presionar el boton “devolver”
  Registra en el sistema, la devolución de un libro que se encontraba en prestamo

```
private void JbDevolver1MouseClicked(java.awt.event.MouseEvent evt) {
    // Verifica si se ha seleccionado una solicitud de devolución
    if (JtaMostrarSolicitudes.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(this, "Por favor, seleccione un usuario con su prestamo que desea devolver.");
        return;
    }

    int selectedRow = JtaMostrarSolicitudes.getSelectedRow();
    String codigoLibro = (String) JtaMostrarSolicitudes.getValueAt(selectedRow, 1);

    // Actualiza el estado del libro en la base de datos
    try {
        connection.setAutoCommit(false);

        // Elimina el préstamo de la tabla 'prestamos'
        try (PreparedStatement ps = connection.prepareStatement("DELETE FROM prestamos WHERE codigo_libro = ?")) {
            ps.setString(1, codigoLibro);
            ps.executeUpdate();
        }

        // Actualiza el estado del libro a 'Disponible' en la tabla 'libros'
        try (PreparedStatement ps = connection.prepareStatement("UPDATE libros SET estado = 'Disponible' WHERE codigo = ?")) {
            ps.setString(1, codigoLibro);
            ps.executeUpdate();
        }

        connection.commit();
    } catch (SQLException e) {
        // Si ocurre un error, se hace rollback a la transacción
        try {
            connection.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        // Restaura el auto-commit al estado original
        try {
            connection.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Recarga las tablas para reflejar los cambios
    cargarTabla();
    mostrarLibrosSolicitados();
}
```