

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2
CATEDRÁTICO: ING. JOSE GRANADOS GUEVARA



ELEAZAR NEFTALÍ COLOP COLOP
CARNÉ: 202131418
SECCIÓN: A

GUATEMALA, 19 DE AGOSTO DE 2025

ÍNDICE

ÍNDICE	1
INTRODUCCIÓN	1
OBJETIVOS	1
1. GENERAL	1
2. ESPECÍFICOS	1
ALCANCES DEL SISTEMA	1
ESPECIFICACIÓN TÉCNICA	1
• REQUISITOS DE HARDWARE	1
• REQUISITOS DE SOFTWARE	1
DESCRIPCIÓN DE LA SOLUCIÓN	2
LÓGICA DEL PROGRAMA	2
➤ Librerías	2
➤ Métodos y Funciones utilizadas	3

INTRODUCCIÓN

Este manual está diseñado para guiar al usuario y al desarrollador en la comprensión y uso de la aplicación desarrollada con tecnología Java, utilizando NetBeans como entorno de desarrollo, MySQL como base de datos, y JDK 21. El propósito de este manual es proporcionar instrucciones claras para que tanto los programadores como los usuarios finales puedan utilizar y modificar la aplicación de manera efectiva.

OBJETIVOS

GENERAL

El objetivo general de este manual es ofrecer una guía completa para la implementación, uso y mantenimiento de la aplicación desarrollada, asegurando que los usuarios y los desarrolladores puedan trabajar eficientemente con la infraestructura técnica utilizada.

ESPECÍFICOS

Objetivo 1: Describir los pasos técnicos que se siguen para configurar el entorno de desarrollo, como lo es MySQL, NetBeans, JDK 21, y la integración entre estos componentes.

Objetivo 2: Explicar el funcionamiento y la estructura del código, dando pinceladas de las clases, funciones y procedimientos que componen la aplicación.

ALCANCES DEL SISTEMA

Este manual está enfocado en la configuración e implementación de un sistema que se conecta a una base de datos MySQL para gestionar información. Como también proporciona la configuración de MySQL y la ejecución de la aplicación en NetBeans. El alcance está limitado a la configuración y funcionamiento del sistema en un entorno de desarrollo basado en Windows, con la versión de JDK 21, y a los componentes utilizados para la gestión de la base de datos el fronted con swing.

ESPECIFICACIÓN TÉCNICA

● REQUISITOS DE HARDWARE

- **Procesador:** 2.0 GHz o superior.
- **Memoria RAM:** 8 GB o más recomendados para el desarrollo de aplicaciones.
- **Espacio en disco:** Al menos 10 GB de espacio libre para la instalación de MySQL y NetBeans.
- **Tarjeta gráfica:** No requerida, a menos que se implementen características visuales intensivas.

● REQUISITOS DE SOFTWARE

- **Sistema Operativo:** Windows 10 o superior.
- **JDK:** JDK 21 para compilar y ejecutar la aplicación Java.
- **MySQL Server:** MySQL 8.0 para gestionar la base de datos.
- **NetBeans IDE:** NetBeans 14 o superior para el desarrollo del código Java y gestión del proyecto.

DESCRIPCIÓN DE LA SOLUCIÓN

La solución propuesta ha sido desarrollada utilizando **Java** con la finalidad de gestionar información de eventos y participantes en un sistema de registro y reportes. La base de datos se gestiona con **MySQL**, donde se almacenan datos de eventos, participantes, asistencias y pagos.

El sistema está construido como una **aplicación de escritorio** utilizando **Swing**, lo que permite ofrecer una interfaz gráfica amigable para el usuario, incluyendo formularios, tablas y menús para la gestión de eventos y generación de reportes en HTML.

La aplicación permite registrar participantes, validar su asistencia, generar reportes de eventos con información de participantes y montos recaudados, y administrar diferentes tipos de eventos como CHARLA, CONGRESO, TALLER o DEBATE.

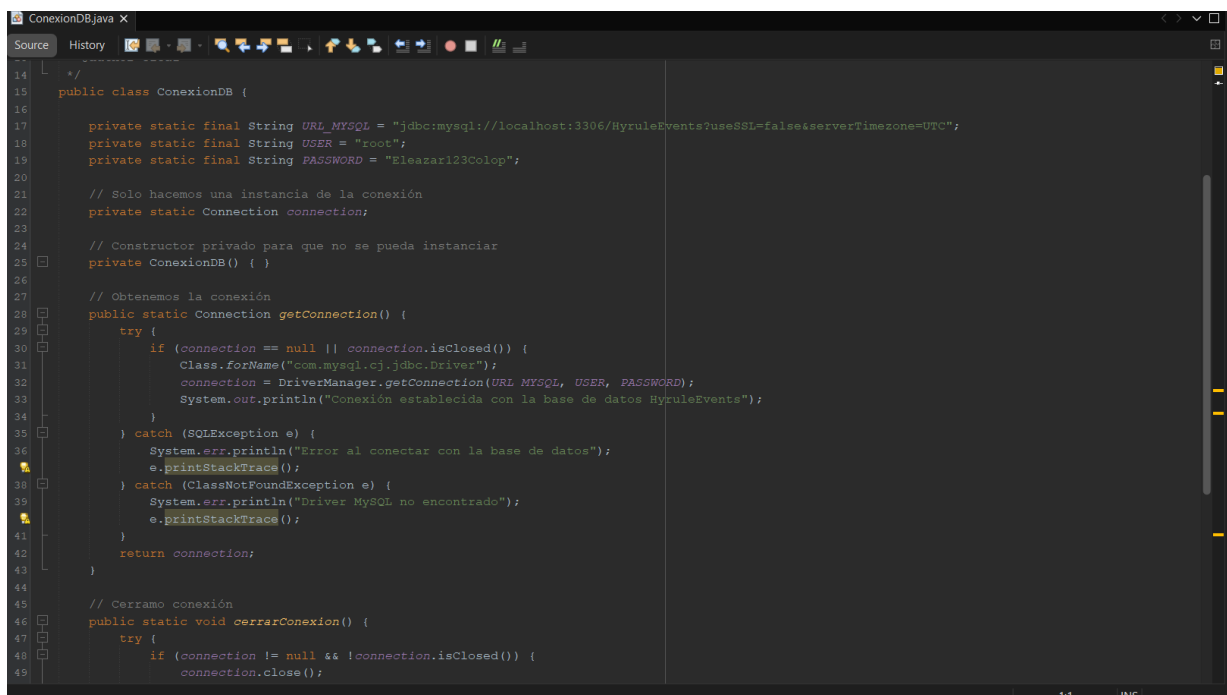
Se ha diseñado la base de datos pensando en la **integridad y consistencia de la información**, así como en la eficiencia de las consultas, permitiendo que las operaciones de registro, modificación y generación de reportes se realicen de forma rápida y segura.

LÓGICA DEL PROGRAMA

La lógica del programa está estructurada para **interactuar directamente con la base de datos** a través de la librería `java.sql.*`. Los componentes principales incluyen:

1. Conexión a la base de datos:

- La clase `Connection` permite establecer la conexión con el servidor MySQL.



```
14  */
15  public class ConexionDB {
16
17      private static final String URL_MYSQL = "jdbc:mysql://localhost:3306/HyruleEvents?useSSL=false&serverTimezone=UTC";
18      private static final String USER = "root";
19      private static final String PASSWORD = "Eleazar123Colop";
20
21      // Solo hacemos una instancia de la conexión
22      private static Connection connection;
23
24      // Constructor privado para que no se pueda instanciar
25      private ConexionDB() { }
26
27      // Obtenemos la conexión
28      public static Connection getConnection() {
29          try {
30              if (connection == null || connection.isClosed()) {
31                  Class.forName("com.mysql.cj.jdbc.Driver");
32                  connection = DriverManager.getConnection(URL_MYSQL, USER, PASSWORD);
33                  System.out.println("Conexión establecida con la base de datos HyruleEvents");
34              }
35          } catch (SQLException e) {
36              System.err.println("Error al conectar con la base de datos");
37              e.printStackTrace();
38          } catch (ClassNotFoundException e) {
39              System.err.println("Driver MySQL no encontrado");
40              e.printStackTrace();
41          }
42          return connection;
43      }
44
45      // Cerramos conexión
46      public static void cerrarConexion() {
47          try {
48              if (connection != null && !connection.isClosed()) {
49                  connection.close();
50              }
51          } catch (SQLException e) {
52              System.err.println("Error al cerrar la conexión");
53              e.printStackTrace();
54          }
55      }
56  }
```

- Las consultas SQL se ejecutan mediante `PreparedStatement`, asegurando la parametrización de los datos y previniendo inyecciones SQL.
- Se manejan posibles errores con `SQLException` para garantizar la estabilidad de la aplicación.


```
InsertarEvento.java X  ConexionDB.java X
Source History
25 public void ingresarEvento(Eventos evento) {
26     String sql = "INSERT INTO evento (codigo_evento, fecha_evento, tipo_evento, titulo, ubicacion, cupo_maximo, precio_evento) "
27         + "VALUES (?, ?, ?, ?, ?, ?, ?)";
28
29     try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
30         pstmt.setString(1, evento.getCodigoEvento());
31
32         // convertimos el LocalDate a java.sql.Date
33         pstmt.setDate(2, java.sql.Date.valueOf(evento.getFechaEvento()));
34
35         pstmt.setString(3, evento.getTipoEvento().name()); // enum guardado como texto
36         pstmt.setString(4, evento.getTituloEvento());
37         pstmt.setString(5, evento.getUbicacion());
38         pstmt.setInt(6, evento.getCupoMaximo());
39         pstmt.setDouble(7, evento.getPrecioEvento());
40
41         int rowsAffected = pstmt.executeUpdate();
42         if (rowsAffected > 0) {
43             System.out.println("Evento ingresado exitosamente.");
44         }
45     } catch (SQLException e) {
46         System.err.println("Error al insertar el evento en la base de datos.");
47         e.printStackTrace();
48     }
49 }
50
51 public boolean existeCodigoEvento(String codigoEvento) {
52     String sql = "SELECT COUNT(*) FROM evento WHERE codigo_evento = ?";
53     try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
54         pstmt.setString(1, codigoEvento);
55         ResultSet rs = pstmt.executeQuery();
56         if (rs.next()) {
57             return rs.getInt(1) > 0;
58         }
59     } catch (SQLException e) {
60         System.err.println("Error al verificar el código del evento.");
61     }
62 }
```

2. Gestión de eventos y participantes:

- La aplicación permite crear eventos y participantes, utilizando formularios en Swing que capturan los datos del usuario.
- Para validar la asistencia de los participantes se realiza un registro en la base de datos y se almacena el estado de validación junto con el método de pago y monto pagado.

```
InsertarEvento.java X  ConexionDB.java X  InsertarPago.java X
Source History
23 public void ingresarPago(Pago pago) throws Exception {
24     // Validamos que exista la inscripción
25     if (!existeInscripcion(pago.getCorreoParticipante(), pago.getCodigoEvento())) {
26         throw new Exception("No existe una inscripción para este participante en el evento indicado.");
27     }
28
29     // Validamos que el monto sea >= precio_evento
30     double precioEvento = obtenerPrecioEvento(pago.getCodigoEvento());
31     if (precioEvento < 0) {
32         throw new Exception("No se encontró el evento asociado al código.");
33     }
34
35     if (pago.getMonto() < precioEvento) {
36         throw new Exception("El monto ingresado no puede ser menor al precio del evento. Precio: " + precioEvento);
37     }
38
39     // Validamos que no exista ya un pago
40     if (existePago(pago.getCorreoParticipante(), pago.getCodigoEvento())) {
41         throw new Exception("Ya existe un pago registrado para este participante en este evento.");
42     }
43
44     // Insertamos el pago
45     String sql = "INSERT INTO pago (correo_participante, codigo_evento, metodo_pago, monto) VALUES (?, ?, ?, ?)";
46     try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
47         pstmt.setString(1, pago.getCorreoParticipante());
48         pstmt.setString(2, pago.getCodigoEvento());
49         pstmt.setString(3, pago.getTipoPago().name());
50         pstmt.setDouble(4, pago.getMonto());
51
52         int rowsAffected = pstmt.executeUpdate();
53         if (rowsAffected > 0) {
54             System.out.println("Pago registrado exitosamente.");
55         }
56
57         // Ahora actualizamos al participante validado en la db
58         actualizarParticipante(pago.getCorreoParticipante());
59     }
60 }
```

3. Generación de reportes:

- Los reportes se generan en formato **HTML**, mostrando la información de cada evento, los participantes, montos totales recaudados y cantidad de participantes validados/no validados.
- La clase ReporteEventos se encarga de consultar la base de datos según filtros (como tipo de evento o código de evento) y construir la página HTML automáticamente.

```

25
26 public void generarReporte(String codigoEvento, String tipoEvento) {
27     String sql = "SELECT e.codigo_evento, e.fecha_evento, e.titulo, e.tipo_evento, "
28         + "e.ubicacion, e.cupo_maximo, p.correo, p.nombre_completo, p.tipo_participante "
29         + "FROM evento e "
30         + "INNER JOIN asistencia a ON e.codigo_evento = a.codigo_actividad "
31         + "INNER JOIN participante p ON a.correo_participante = p.correo "
32         + "WHERE e.codigo_evento = ? AND e.tipo_evento = ?";
33
34     try (PreparedStatement ps = conexion.prepareStatement(sql)) {
35         ps.setString(1, codigoEvento);
36         ps.setString(2, tipoEvento);
37
38         ResultSet rs = ps.executeQuery();
39
40         // Crear carpeta reportes/eventos si no existe
41         File carpeta = new File("reportes/eventos");
42         if (!carpeta.exists()) {
43             carpeta.mkdirs();
44         }
45
46         String nombreArchivo = "reportes/eventos/reporte_" + codigoEvento + ".html";
47
48         StringBuilder html = new StringBuilder();
49         html.append("<html><body style='font-family:Arial, sans-serif;'>");
50         html.append("<h2>REPORTE DE EVENTO</h2>");
51
52         html.append("<table border='0' cellspacing='0' cellpadding='5'>");
53         html.append("<tr><td><b>CÓDIGO DE EVENTO:</b></td><td></td></tr>");
54         html.append("<tr><td><b>FECHA DE EVENTO:</b></td><td></td></tr>");
55         html.append("<tr><td><b>TÍTULO DEL EVENTO:</b></td><td></td></tr>");
56         html.append("<tr><td><b>TIPO DE EVENTO:</b></td><td></td></tr>");
57         html.append("<tr><td><b>UBICACIÓN:</b></td><td></td></tr>");
58         html.append("<tr><td><b>CUPO MÁXIMO:</b></td><td></td></tr>");
59         html.append("</table><br>");
60

```

4. Interfaz de usuario:

- La interfaz se desarrolla completamente con **Swing**, utilizando JFrame, JPanel, JTable, JButton, JTextField y JComboBox.
- Se busca que la experiencia del usuario sea intuitiva: los formularios permiten ingresar datos, seleccionar opciones y ejecutar acciones como generar reportes con un solo clic.

5. Separación de responsabilidades:

- Se mantiene una estructura clara entre la **lógica de negocio** (consultas SQL, validaciones y cálculos) y la **interfaz de usuario** (formularios y visualización de datos).

- Esto facilita el mantenimiento, la escalabilidad y la incorporación de nuevas funcionalidades en el futuro.

Ejemplo de código:

conexión a db:

```
public class ConexionDB {

    private static final String URL_MYSQL =
"jdbc:mysql://localhost:3306/HyruleEvents?useSSL=false&serverTimezone=UTC";

    private static final String USER = "root";
    private static final String PASSWORD = "Eleazar123Colop";

    // Solo hacemos una instancia de la conexión
    private static Connection connection;

    // Constructor privado para que no se pueda instanciar
    private ConexionDB() { }

    // Obtenemos la conexión
    public static Connection getConnection() {
        try {
            if (connection == null || connection.isClosed()) {
                Class.forName("com.mysql.cj.jdbc.Driver");
                connection = DriverManager.getConnection(URL_MYSQL, USER,
PASSWORD);
                System.out.println("Conexión establecida con la base de datos
HyruleEvents");
            }
        } catch (SQLException e) {
            System.err.println("Error al conectar con la base de datos");
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            System.err.println("Driver MySQL no encontrado");
            e.printStackTrace();
        }
    }
}
```

```

        return connection;
    }

    // Cerramo conexión
    public static void cerrarConexion() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
                System.out.println("Conexión cerrada");
            }
        } catch (SQLException e) {
            System.err.println("Error al cerrar la conexión");
            e.printStackTrace();
        }
    }
}

```

Insertar algo el la db:

```

public class InsertarEvento {

    private Connection connection;

    public InsertarEvento(Connection connection) {
        this.connection = connection;
    }

    public void ingresarEvento(Eventos evento) {
        String sql = "INSERT INTO evento (codigo_evento, fecha_evento,
        tipo_evento, titulo, ubicacion, cupo_maximo, precio_evento) "
            + "VALUES (?, ?, ?, ?, ?, ?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, evento.getCodigoEvento());

            // convertimos el LocalDate a java.sql.Date
            pstmt.setDate(2, java.sql.Date.valueOf(evento.getFechaEvento()));

```

```

        pstmt.setString(3, evento.getTipoEvento().name()); // enum guardado
como texto
        pstmt.setString(4, evento.getTituloEvento());
        pstmt.setString(5, evento.getUbicacion());
        pstmt.setInt(6, evento.getCupoMaximo());
        pstmt.setDouble(7, evento.getPrecioEvento());

        int rowsAffected = pstmt.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Evento ingresado exitosamente.");
        }
    } catch (SQLException e) {
        System.err.println("Error al insertar el evento en la base de datos.");
        e.printStackTrace();
    }
}

public boolean existeCodigoEvento(String codigoEvento) {
    String sql = "SELECT COUNT(*) FROM evento WHERE codigo_evento =
?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString(1, codigoEvento);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt(1) > 0;
        }
    } catch (SQLException e) {
        System.err.println("Error al verificar el código del evento.");
        e.printStackTrace();
    }
    return false;
}
}

```

Pequeña explicación:

- **PreparedStatement**

- Es como una plantilla para escribir consultas SQL.
- Te permite poner “huecos” (?) que luego se llenan con datos reales.

- Ayuda a que **no se pueda hacer trampa** (inyección SQL) y hace que las consultas sean más seguras.
- **ResultSet**
 - Es como una tabla de resultados que devuelve la base de datos después de hacer una consulta (SELECT).
 - Te permite **leer fila por fila** los datos que pediste.
- **Connection**
 - Es la conexión entre tu programa y la base de datos.
 - Sin esto, tu programa no podría leer ni guardar datos.
- **executeUpdate()**
 - Es el método que **ejecuta instrucciones que cambian la base de datos** (como INSERT, UPDATE o DELETE).
 - Devuelve cuántas filas se afectaron.
- **executeQuery()**
 - Es el método que **ejecuta consultas que solo leen datos** (SELECT).
 - Devuelve un `ResultSet` con los resultados.
- **try-with-resources**
 - Es una forma de **abrir recursos (como conexiones o consultas) y cerrarlos automáticamente** cuando ya no se necesitan.
 - Evita que queden cosas abiertas y que tu programa tenga errores o fugas de memoria.
- **SQLException**
 - Es un tipo de error que ocurre cuando **algo sale mal al trabajar con la base de datos**.
 - Puede pasar si la tabla no existe, si los datos son incorrectos, o si la conexión falla.

Insertar desde el botón:

```
try {
    // Extraemos valores
    String codigoEvento = jTextField1.getText().trim();
    String tituloEvento = jTextField2.getText().trim();
    String ubicacion = jTextArea1.getText().trim();
    String cupoStr = jTextField3.getText().trim();
    String precioStr = jTextField4.getText().trim(); // nuevo campo

    // Validamos campos vacíos
    if (codigoEvento.isEmpty() || tituloEvento.isEmpty() ||
    ubicacion.isEmpty() || cupoStr.isEmpty() || precioStr.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Todos los campos son
obligatorios.");
        return;
    }

    // Validamos fecha
    java.util.Date fechaUtil = jDateChooser1.getDate();
    if (fechaUtil == null) {
        JOptionPane.showMessageDialog(this, "Debe seleccionar una fecha
para el evento.");
        return;
    }
    java.sql.Date fechaSql = new java.sql.Date(fechaUtil.getTime());
    LocalDate fechaEvento = fechaSql.toLocalDate();

    // Validamos tipo de evento
    String tipoSeleccionado = (String) jComboBox3.getSelectedItem();
    TipoEvento tipoEvento =
TipoEvento.valueOf(tipoSeleccionado.toUpperCase());

    // Validamos ubicación
    if (ubicacion.length() > 150) {
        JOptionPane.showMessageDialog(this, "La ubicación no puede superar
los 150 caracteres.");
        jTextArea1.setText("");
        return;
    }
}
```

```

// Validamos cupo
int cupoMaximo;
try {
    cupoMaximo = Integer.parseInt(cupoStr);
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Error: El cupo máximo debe
ser un número válido.");
    return;
}

// Validamos precio
double precioEvento;
try {
    precioEvento = Double.parseDouble(precioStr);
    if (precioEvento < 0) {
        JOptionPane.showMessageDialog(this, "El precio del evento no
puede ser negativo.");
        return;
    }
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Error: El precio del evento
debe ser un número válido.");
    return;
}

// aca funca la conexión e inserción
InsertarEvento insertar = new
InsertarEvento(ConexionDB.getConnection());

// Validamos código único
if (insertar.existeCodigoEvento(codigoEvento)) {
    JOptionPane.showMessageDialog(this, "El código de evento ya existe.
Ingrese uno diferente.");
    jTextField1.setText("");
    return;
}

// Creamos y guardar evento

```



```

Eventos nuevoEvento = new Eventos(
    codigoEvento,
    fechaEvento,
    tipoEvento,
    ubicacion,
    tituloEvento,
    cupoMaximo,
    precioEvento
);
insertar.ingresarEvento(nuevoEvento);

JOptionPane.showMessageDialog(this, "Evento registrado
exitosamente.");

// Limpiamos campos
jTextArea1.setText("");
jTextField1.setText("");
jTextField2.setText("");
jTextField3.setText("");
jTextField4.setText("");
jDateChooser1.setDate(null);
jComboBox3.setSelectedIndex(0);

} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Error al registrar el evento: " +
e.getMessage());
    e.printStackTrace();
}

```

Conceptos clave del código del botón de registro de eventos

1. trim()

- Se usa en todos los campos de texto, por ejemplo:
- `String codigoEvento = jTextField1.getText().trim();`
- **Qué hace:** elimina los espacios al principio y al final del texto.
- **Por qué es importante:** evita errores si el usuario pone espacios sin darse cuenta, por ejemplo " EVT001 " se convierte en "EVT001".

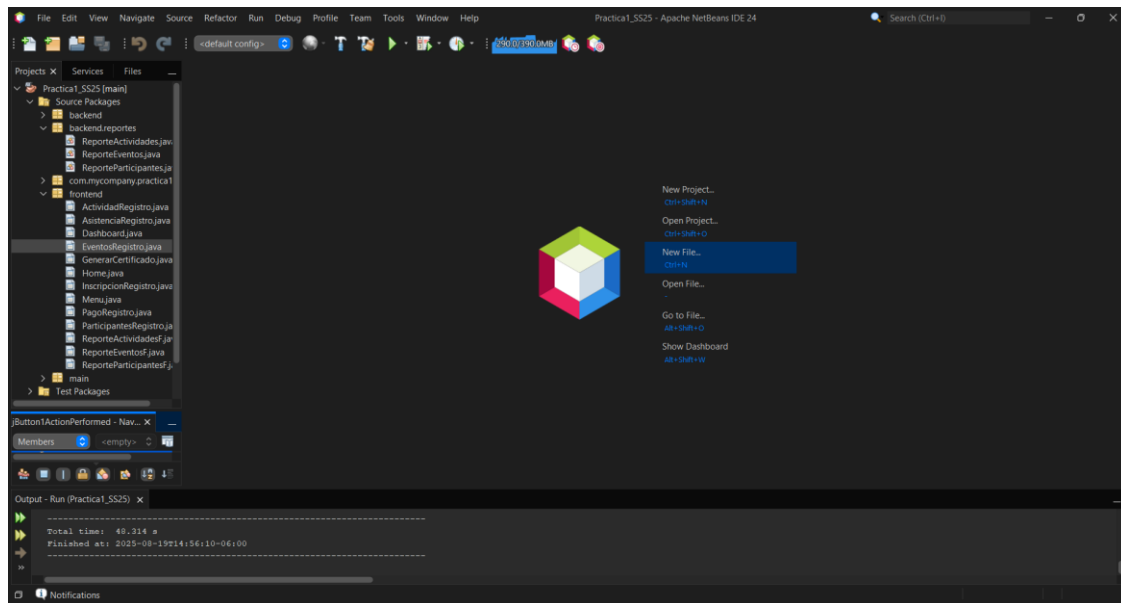
2. Validaciones de campos vacíos

- Se revisa que todos los campos tengan algún valor antes de guardar.
- Evita que se ingresen eventos incompletos.

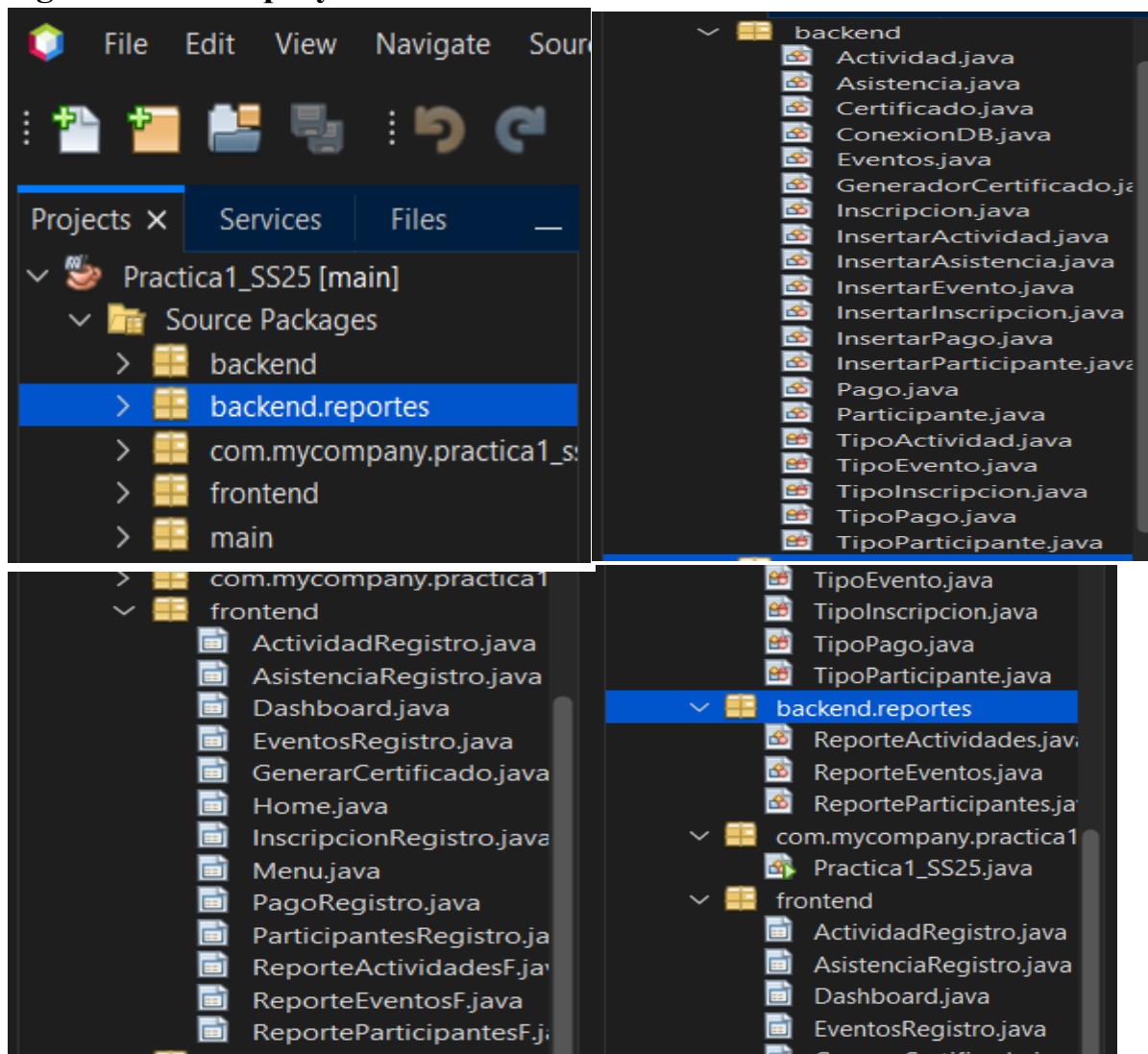
3. Validación de fecha

- Se obtiene la fecha seleccionada en el JDateChooser y se convierte a LocalDate.
 - Se asegura que **el evento tenga una fecha válida**.
- 4. Validación de tipo de evento**
- Convierte la selección del ComboBox a un valor del enum TipoEvento.
 - Asegura que el tipo del evento esté entre los permitidos (CHARLA, CONGRESO, etc.).
- 5. Validación de ubicación**
- Verifica que la ubicación no supere 150 caracteres, según lo definido en la base de datos.
- 6. Validación de cupo máximo**
- Convierte el texto ingresado a número entero (int).
 - Si no es un número válido, muestra un mensaje de error.
- 7. Validación de precio**
- Convierte el texto a número decimal (double) y verifica que no sea negativo.
 - Asegura que el precio sea válido antes de guardarlo.
- 8. Validación de código único**
- Antes de insertar, se revisa si el código del evento ya existe en la base de datos.
 - Evita duplicados que podrían causar errores o confusión.
- 9. Inserción en la base de datos**
- Se crea un objeto Eventos con los datos ingresados.
 - Se llama al método ingresarEvento de la clase InsertarEvento para guardar el evento en la base de datos.
- 10. Limpieza de campos**
- Después de guardar, se borran todos los campos de entrada para que el formulario quede listo para un nuevo registro.
- 11. Manejo de errores (try-catch)**
- Si ocurre algún error durante el proceso, se muestra un mensaje al usuario y se imprime la excepción para depuración.

Anexos: netbeans:



organizacion del proyecto:



Ingresar a db desde terminal:

```
PowerShell
PowerShell 7.5.2
PS C:\Users\eleaz> cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
PS C:\Program Files\MySQL\MySQL Server 8.0\bin> .\mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 175
Server version: 8.0.41 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use hyruleevents;
Database changed
mysql> |
```

tablas:

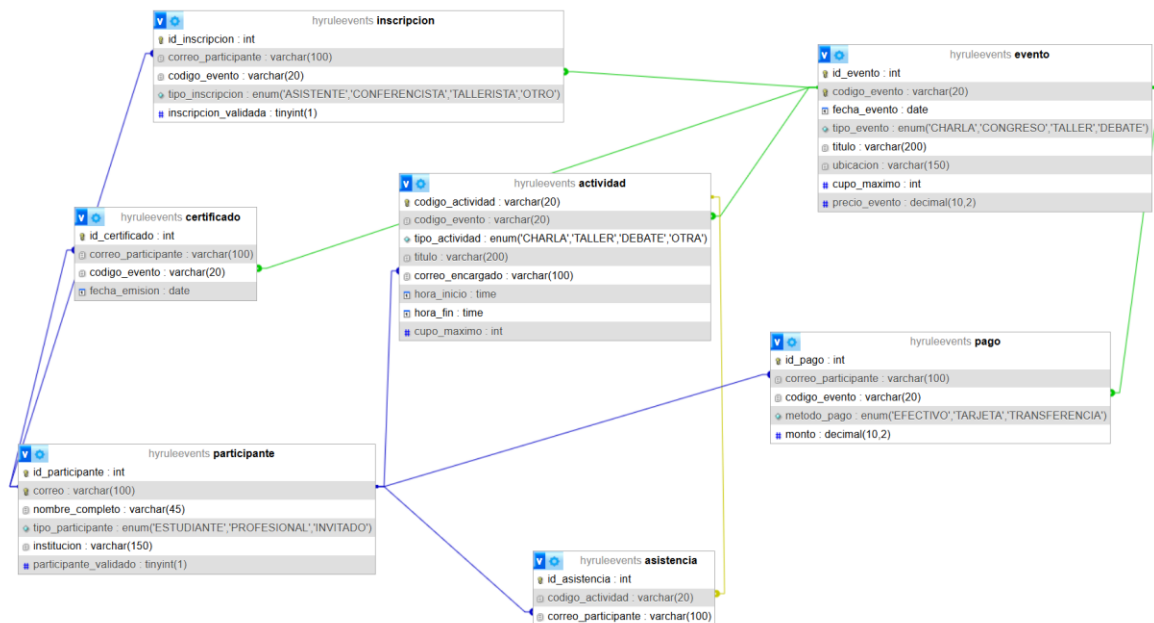


diagrama entidad relación:

