

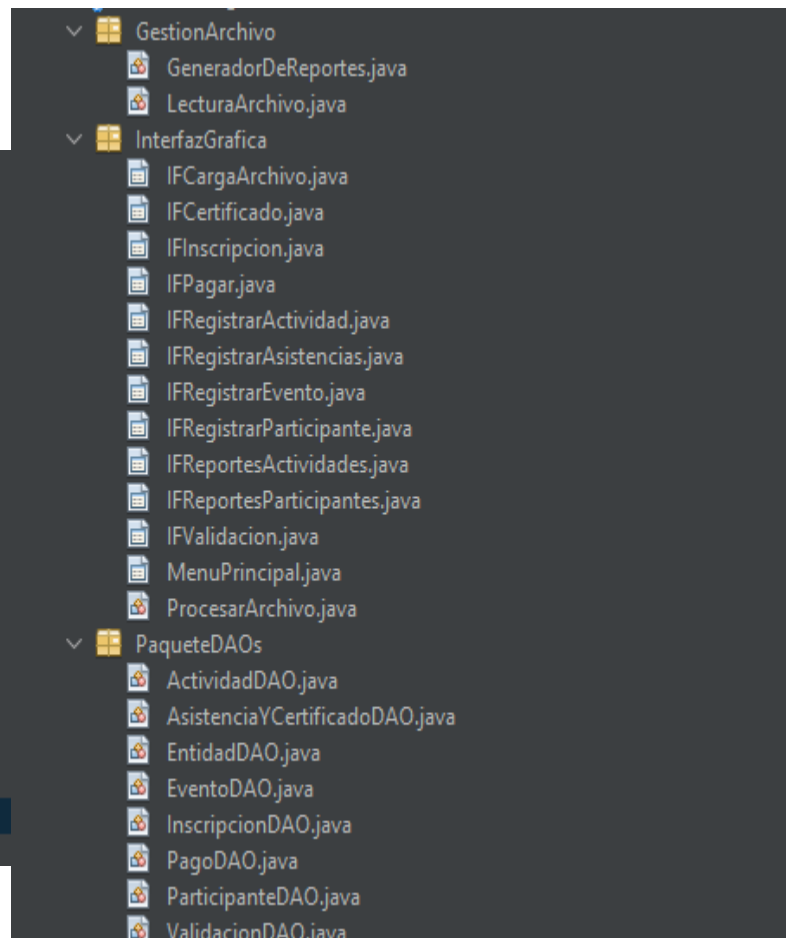
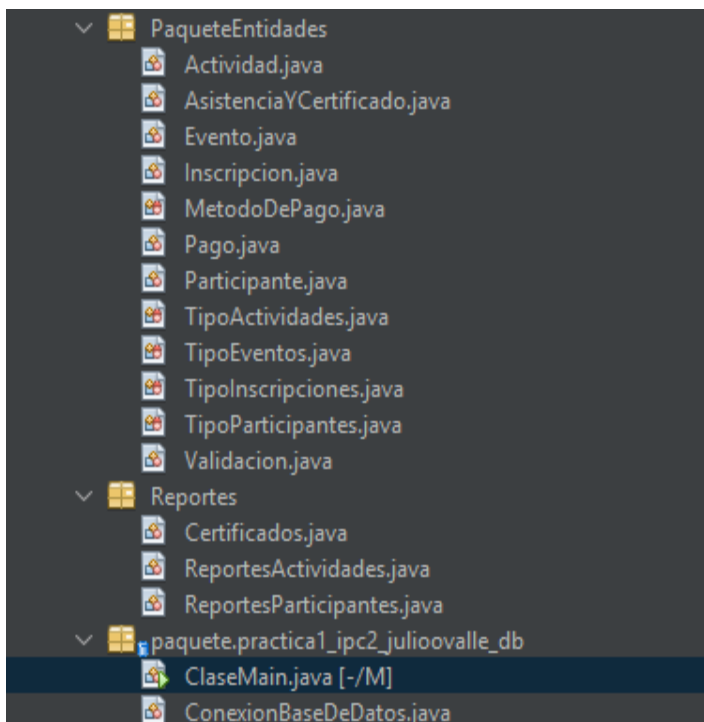
Manual técnico de “Aplicación para control de eventos y actividades en el Reino de Hyrule”

Bienvenido al Manual Técnico de la aplicación de Triforce Software para el manejo de eventos en el reino de Hyrule

Lenguaje de programación e IDE utilizados en el proyecto:

- Java, versión 21.0.04
- IDE: Apache NetBeans 20

Conjunto de clases utilizadas:



Método main:

```
package paquete.practical_ipc2_julioovalle_db;
import InterfazGrafica.MenuPrincipal;
import java.sql.*;
import java.util.Scanner;

public class ClaseMain {

    public static void main(String[] args) {

        Connection conn = ConexionBaseDeDatos.conectarConBaseDeDatos();//Se conecta con la base de datos

        Scanner escaner = new Scanner(System.in);

        MenuPrincipal menu = new MenuPrincipal(conn);
        menu.setVisible(true);

    }
}
```

Clase encargada de la conexión con la base de datos:

```
package paquete.practical_ipc2_julioovalle_db;

import java.sql.*;

public class ConexionBaseDeDatos {

    private static final String IP = "localhost";
    private static final int PUERTO = 3306;
    private static final String SCHEMA = "eventos_hyrule";
    private static final String USER = "universal";
    private static final String PASSWORD = "12345";

    private static final String URL = "jdbc:mysql://" + IP + ":" + PUERTO + "/" + SCHEMA;

    private static Connection conect;

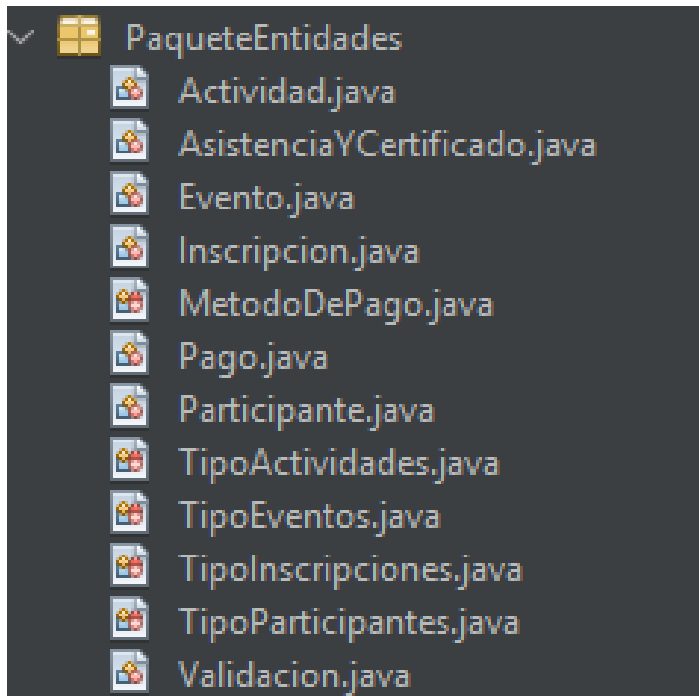
    public static Connection conectarConBaseDeDatos() {

        try {
            conect = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Base de Datos:" + conect.getSchema());
            System.out.println("Catalogo: " + conect.getCatalog());
            return conect;
        } catch (SQLException e) {
            System.out.println("Error al conectarse a la BD");
            e.printStackTrace();
            return null;
        }
    }
}
```

Se necesita un usuario y una contraseña

- Usuario:
universal
- Contraseña
: 12345

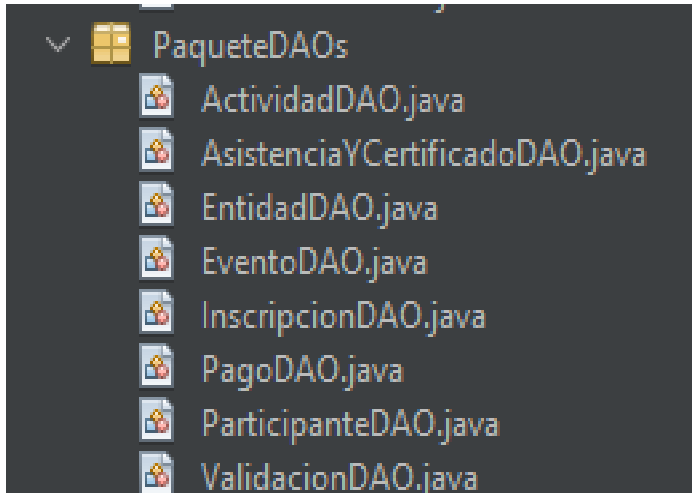
Paquete de clases “Entidades”:



En este paquete se encuentran las clases que representan a todas las entidades en la base de datos relacional en la que se basa la aplicación. Cada clase cuenta con sus atributos, sus constructores y sus métodos para obtener y colocar valores a los atributos si así lo requiere el programa

```
public class Actividad {  
  
    private String codigoActividad, codigoEvento, eMailEncargado, Titulo;  
    private TipoActividades tipoActividad;  
    private int cupoMaximo;  
    private LocalTime horaInicio, horaFin;  
  
    public Actividad(String codigoActividad, String codigoEvento, String eMailEncargado, String Titulo, TipoActividades tipoActividad,  
        int cupoMaximo, LocalTime horaInicio, LocalTime horaFin) {  
        this.codigoActividad = codigoActividad;  
        this.codigoEvento = codigoEvento;  
        this.eMailEncargado = eMailEncargado;  
        this.Titulo = Titulo;  
        this.tipoActividad = tipoActividad;  
        this.cupoMaximo = cupoMaximo;  
        this.horaInicio = horaInicio;  
        this.horaFin = horaFin;  
    }  
  
    public String getCodigoActividad() {  
        return codigoActividad;  
    }  
  
    public void setCodigoActividad(String codigoActividad) {  
        this.codigoActividad = codigoActividad;  
    }  
}
```

Paquete de clases “DAO”:



En este paquete se encuentran las clases que representan a todas las conexiones de las entidades con la base de datos, DAO (Data Access Object). En estas clases se encuentran todos los métodos y funciones que se ocupan de hacer las búsquedas y/o realizar alteraciones en sus respectivas tablas. Cada clase DAO necesita una conexión “Connection conn”

Ejemplo en clase “ActividadDAO”:

```
public void comprobarExistencia(Actividad actividad) {  
  
    //Si no hay una actividad con el mismo codigo,  
    //Si el correo del encargado existe  
    //Y si el evento existe se podrá registrar la actividad  
  
    if (!(buscarPorParametros(actividad.getCodigoActividad(), "codigo_actividad", "actividad", getConn()) && buscarPorParametros(actividad.getCodigoEvento(), "codigo_evento", "evento", getConn()) {  
  
        if (comprobarEncargado(actividad.getMailEncargado())) { //Si cumple los parametros, crea la actividad  
            registrarActividad(actividad);  
        }  
  
    } else {  
        System.out.println("Los datos no coinciden");  
        JOptionPane.showMessageDialog(null, "Datos incoherentes, revise sus datos", "ADVERTENCIA", JOptionPane.WARNING_MESSAGE);  
    }  
}
```

método que se encarga de asegurarse que los datos que ingreso el usuario existan

```

public void registrarActividad(Actividad actividad) { //Registra el actividad en la BD

    String insertarActividad = "INSERT INTO actividad (codigo_actividad, codigo_evento, email_encargado, "
        + "titulo,tipo_actividad, cupo_maximo,hora_inicio, hora_fin) "
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?) ";

    try {
        PreparedStatement ps = getConn().prepareStatement(insertarActividad);

        ps.setString(1, actividad.getCodigoActividad());
        ps.setString(2, actividad.getCodigoEvento());
        ps.setString(3, actividad.getMailEncargado());
        ps.setString(4, actividad.getTitulo());
        ps.setString(5, actividad.getTipoActividad().name());
        ps.setInt(6, actividad.getCupoMaximo());
        ps.setTime(7, java.sql.Time.valueOf(actividad.getHoraInicio()));
        ps.setTime(8, java.sql.Time.valueOf(actividad.getHoraFin()));

        int n = ps.executeUpdate();
        System.out.println("sql ejecutado: " + ps);
        System.out.println("Rows affected " + n);
        JOptionPane.showMessageDialog(null, "Actividad registrada con exito", "Todo bien", JOptionPane.PLAIN_MESSAGE);

    } catch (SQLException e) {
        //e.printStackTrace(); //lo comento porque el erro que imprime es muy grande y parece que el programa falla
        System.out.println("Error al registrar Actividad");
    }
}

```

método que se encarga de registrar una actividad en la BD después de pasar todos los filtros para asegurarse que los datos sean validos

Ejemplo en clase PagoDAO:

```

public void verificaPagoSuficiente(Pago pago, double monto) { //antes de terminar verifica si el monto es suficiente
    String pregunta = "SELECT costo_inscripcion FROM evento WHERE codigo_evento = ?";

    try {
        PreparedStatement ps = getConn().prepareStatement(pregunta);
        ps.setString(1, pago.getCodigo_evento());
        ResultSet rs = ps.executeQuery();

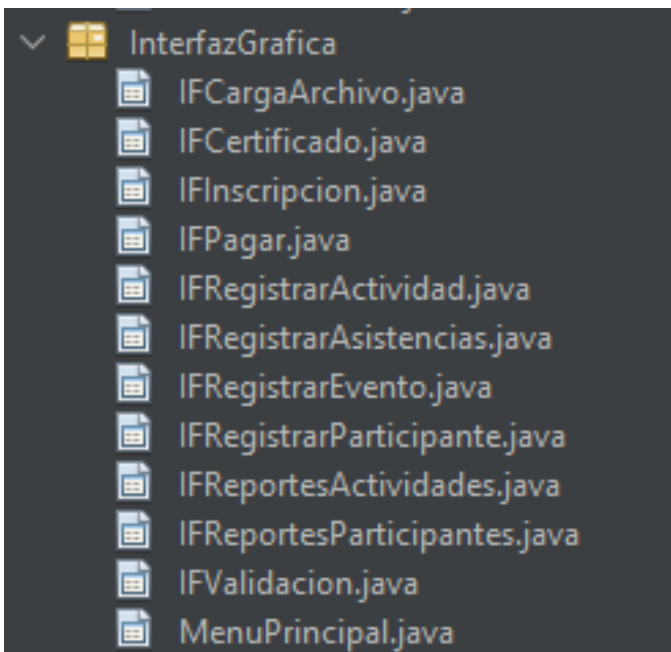
        System.out.println("SQL ejecutado " + ps);

        if (rs.next()) {
            double costoEvento = rs.getDouble("costo_inscripcion");
            if (monto >= costoEvento) { //Si el monto recibido es por lo menos el esperado
                pagarInscripcion(pago, monto);
            } else {
                JOptionPane.showMessageDialog(null, "monto insuficiente", "ERROR", JOptionPane.ERROR_MESSAGE);
            }
        }
    } catch (Exception e) {
        System.out.println("Erro en revisar el monto");
    }
}

```

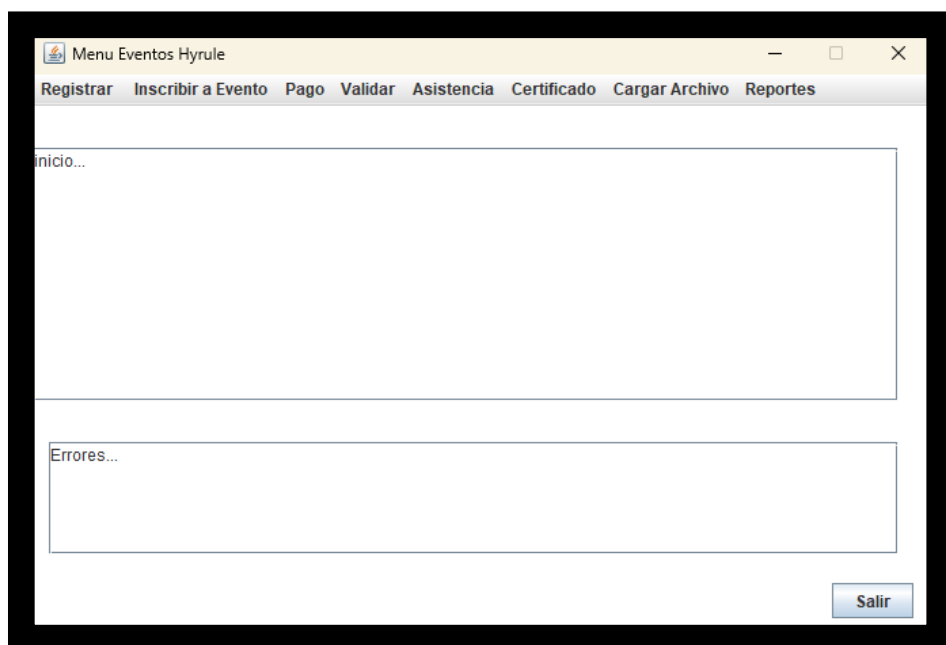
método que se encarga de verificar que el monto que el usuario ingreso para pagar se por lo menos el necesario e indicado en el costo de inscripción de cada evento

Paquete de clases “InterfazGrafica”




En este paquete se encuentran las clases que se encargan de toda la parte que el usuario verá, en estas clases se podrán encontrar con el menú principal y todos los formularios. Estas clases también necesitan una conexión con la BD.

Menú Principal:



Registrar Evento:

Registrar Inscribir a Evento Pago Validar Asistencia Certificado Cargar

 Registrar Evento

Codigo

Fecha (aa-mm-dd)

Titulo

Cupo Maximo

Ubicacion


Costo de Inscripcion

CHARLA

▼

Registrar

Carga de Archivo:



Archivo de entrada

Buscar

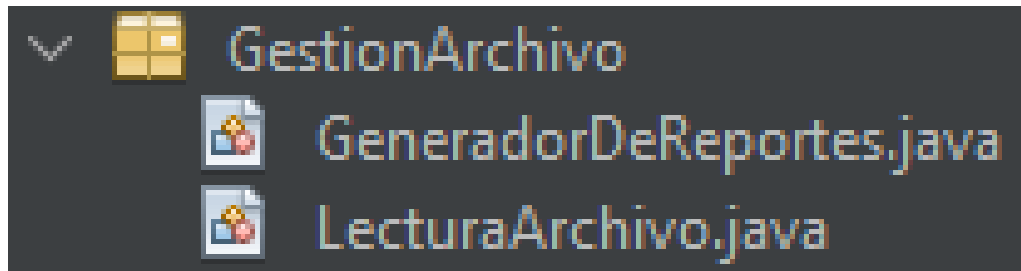
Velocidad de procesamiento (milisegundos)

Archivo de salida

Buscar

Aceptar

Paquete de clases” GestionArchivos”:



En este paquete se encuentran las clases que se encargan de toda la parte de la lectura, interpretación del archivo de entrada y generación de los reportes en formato HTML.

Clase “GestorDeReportes”:

```
public List<Certificados> obtenerCertificados() {  
    //Lista en donde iran los objetos que se mostraran en los reportes  
    List<Certificados> lista = new ArrayList<>();  
    TipoParticipantes tipo = TipoParticipantes.ESTUDIANTE; //por defecto  
    String sql = "SELECT email_participante, codigo_evento FROM certificado";  
  
    System.out.println(sql);  
  
    try (PreparedStatement ps = conn.prepareStatement(sql)) {  
        ResultSet rs = ps.executeQuery();  
  
        while (rs.next()) {  
            Certificados cert = new Certificados(  
                rs.getString("email_participante"),  
                rs.getString("codigo_evento")  
            );  
            lista.add(cert);  
        }  
    } catch (SQLException ex) {  
        JOptionPane.showMessageDialog(null, "Error en los datos", "ERROR", JOptionPane.ERROR_MESSAGE);  
        ex.printStackTrace();  
    }  
    return lista;  
}
```

Función que se ocupa de hacer la consulta para saber cuantos y cuales son los certificados registrados en la BD y los encierra en una ArrayList


```

public void generarReporteCertificados( String rutaDestino) {

    List<Certificados> certificados = obtenerCertificados();

    StringBuilder html = new StringBuilder();

    html.append("<html><head><title>Certificados</title></head><body>");
    html.append("<h1>Certificados</h1>");
    html.append("<table border='1'>");
    html.append("<tr><th>Email_participante</th><th>Codigo_evento</th></tr>");

    for (Certificados c : certificados) {
        html.append("<tr>")
            .append("<td>").append(c.getEmail_participante()).append("</td>")
            .append("<td>").append(c.getCodigo_evento()).append("</td>")
            .append("</tr>");
    }

    html.append("</table>");
    html.append("</body></html>");

    try {
        FileWriter writer = new FileWriter(rutaDestino + "/Certificados.html");
        writer.write(html.toString());
        writer.close();
        System.out.println("Reporte generado en: " + rutaDestino);
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

```

Método que se encarga de realizar el archivo HTML con los reportes. Recibe el ArrayList con los objetos para obtener sus datos y la ruta en donde se encuentra la carpeta para guardar los reportes

Clase “LecturaArchivo”:

```
@Override
public void run() { //Carga el archivo

    try {

        FileReader fr = new FileReader(archivo);
        BufferedReader br = new BufferedReader(fr);

        String linea;
        System.out.println("");
        System.out.println("Contenido del archivo:");

        // Leer línea por línea
        while ((linea = br.readLine()) != null) {

            String texto = linea;
            leerArchivo(linea);

            SwingUtilities.invokeLater(() -> {
                menu.getTextArea().append(texto + "\n");
            });

            Thread.sleep(velocidad); // espera según la velocidad que asigna el usuario
        }

        // Cerrar el lector
        br.close();
        fr.close();

    } catch (IOException e) {
        System.out.println("Error al leer el archivo: " + e.getMessage());
    } catch (InterruptedException er) {
        System.out.println("El hilo se detuvo");
    }

}
```

Esta clase implementa la interfaz “Runnable” la cual permite hacer de esta un hilo el cual ayudará al momento de leer el archivo. El método run lee el archivo línea por línea y la envía a otro método el cual se ocupa de interpretarla

```

public void leerArchivo(String linea) {

    if (linea.startsWith("REGISTRO_EVENTO")) {
        String[] valores = extraerParametros(linea);
        registrarEvento(valores, linea);
    } else if (linea.startsWith("REGISTRO_PARTICIPANTE")) {

        String[] valores = extraerParametros(linea);
        registrarParticipante(valores, linea);
    } else if (linea.startsWith("INSCRIPCION")) {
        String[] valores = extraerParametros(linea);
        inscribir(valores, linea);
    } else if (linea.startsWith("PAGO")) {
        String[] valores = extraerParametros(linea);
        pagar(valores, linea);
    } else if (linea.startsWith("VALIDAR_INSCRIPCION")) {
        String[] valores = extraerParametros(linea);
        validar(valores, linea);
    } else if (linea.startsWith("REGISTRO_ACTIVIDAD")) {
        String[] valores = extraerParametros(linea);
        registrarActividad(valores, linea);
    }
}

```

Este método interpreta las líneas del archivo al identificar el comienzo de cada una y así saber cual de todos los métodos llamar

```

private void registrarEvento(String[] valores, String linea) {

    try {

        int cupo = Integer.parseInt(valores[5]);
        LocalDate fecha = LocalDate.parse(valores[1]);
        TipoEventos tipo = TipoEventos.CHARLA; //por defecto
        tipo = tipo.valueOf(valores[2]);
        double costo = Double.parseDouble(valores[6]);

        Evento nuevoEvento = new Evento(valores[0], valores[3], valores[4], cupo, fecha, tipo, costo);
        EventoDAO even = new EventoDAO(conn);
        even.comprobarExistencia(nuevoEvento);

    } catch (Exception e) {
        System.out.println("Error en la linea: " + linea);
        menu.getTxtAreaErrores().append("\n error en la linea --> " + linea);
    }
}

```

Este método recibe todos los datos que ya fueron identificados previamente y se comunica con las clases DAO para realizar los cambios en la BD

Diagrama E/R:

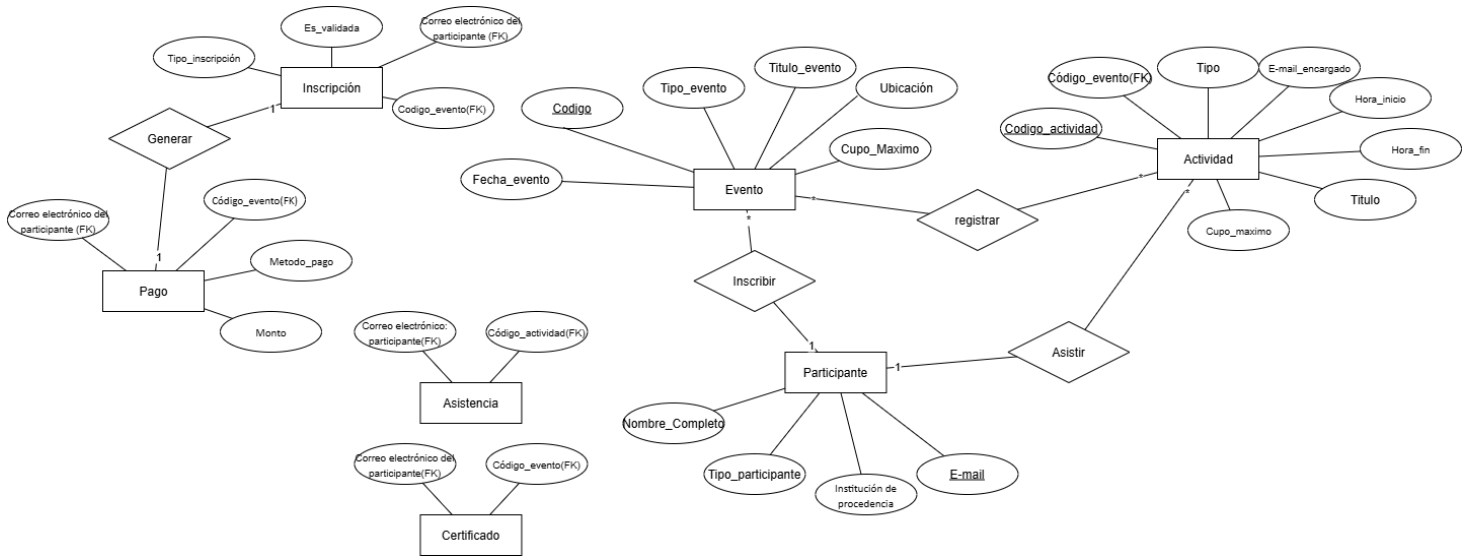


Diagrama de Clases UML:

