

MANUAL TÉCNICO DE MANGO GAMES



Manual realizado por: *Álvaro Carrasco García,*
David Bejarano Llano, José Manuel Rodríguez Cortés

Tabla de contenido

Tabla de contenido	2
Introducción:.....	4
Entorno de desarrollo necesario:	4
Contenido del proyecto	5
Las partes del proyecto están divididas entre:	5
-CSV:.....	5
-Images:.....	5
-Pagina_Home:	6
-Reportes:	6
Documentación de las clases de proyecto	6
1.homeMain.....	7
2. consultas_sql.....	8
Resumen de la clase:	8
Constructor:	8
Métodos:.....	8
Insertar_fichero_csv ():.....	9
IsParsable():	11
Parsear_cadena():	11
Parsear_atributos():	11
3.homeInterface	12
Resumen de la clase:	12
Constructor:	13
Métodos:.....	13
esAdmin():.....	14
busquedaCategorias():.....	14
4.inicio_sesion	15
Constructor:	16
Métodos:.....	16
5.InterfazJuego	17
Resumen de la clase:	17
Constructor:	17
Métodos:.....	18

Convertir_Imagen():.....	18
ComprobarInicioS():	18
esAdmin():.....	19
Boton_comprarActionPerformed():	19
6.registro.....	20
Resumen de la clase:	20
Constructor:	20
Métodos:.....	21
borrar_Campos():.....	21
ComprobarInicioS():	21
esAdmin():.....	21
RegistroActionPerformed():.....	22
7.Registro_juegos	22
Resumen de la clase:	23
Constructor:	23
Métodos:.....	23
borrar_Campos():.....	23
ComprobarInicioS():	24
esAdmin():.....	24
EcreacionInformes():.....	24
8.Juego	26
Resumen de la clase:	26
Constructor:	26
Métodos:.....	26
9.Usuario	26
Resumen de la clase:	26
Constructor:	26
Métodos:.....	27

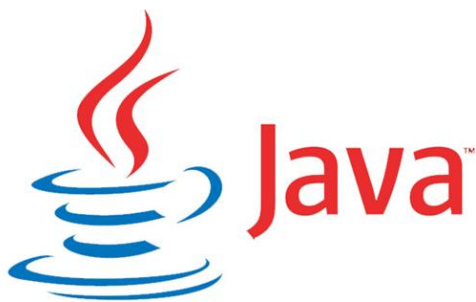
Introducción:

Este proyecto se ha realizado en la asignatura de *Desarrollo de Interfaces* por alumnos de 2º del ciclo formativo *Desarrollo de Aplicaciones Multiplataforma*.

Este proyecto consiste en una simulación de una plataforma de compra de juegos online, similar a Steam, con funcionalidades de introducir juegos y usuarios nuevos en la base de datos utilizada, “compras” de juegos que quedarán registradas junto con el usuario que lo adquiera, búsqueda de juegos mediante palabras clave además de diversas formas de acceder a una página HTML cuya función es la de manual de usuario, para resolver las dudas que puedan sucederle a un usuario sin conocimientos técnicos.

Este manual, por otra parte, tiene la intención de ofrecer ayuda técnica a otros usuarios que utilicen el programa y tengan mayores conocimientos de cómo funciona el código, ofreciendo descripciones detalladas de las diferentes clases, su funcionamiento, sus métodos y los parámetros que estos necesitan, todo con un gran cuidado al detalle y ofreciendo un acceso rápido desde la tabla de contenidos detallada en la página superior.

Entorno de desarrollo necesario:



Para desarrollar la aplicación se ha usado el **IDE Netbeans**, ya que tiene la capacidad de generar interfaces gráficas gracias a las librerías de **JavaSwing**. Esta es una librería capaz de generar interfaces de usuario mediante la lectura de documentos **XML**, este documento se generará a medida que vayamos modificando la interfaz gracias a las herramientas que nos proporciona esta librería.



Para la base de datos se ha utilizado un servidor **MySQL** dentro de un contenedor en **Docker**. Este se inicializará en el propio dispositivo en el **puerto 3306**. Para esto se requerirá la versión más reciente de Docker y una imagen de un servidor MySQL. La imagen la proporcionaremos a partir de este enlace:

https://hub.docker.com/_/mysql

El resto de instrucciones están descritas en el archivo **"readme.txt"**

Contenido del proyecto

Nombre	Fecha de modificación	tipo	tamaño
csv	02/02/2023 12:51	Carpeta de archivos	
help	02/02/2023 12:51	Carpeta de archivos	
Images	02/02/2023 12:51	Carpeta de archivos	
org	05/12/2022 14:06	Carpeta de archivos	
Pagina_Home	02/02/2023 12:51	Carpeta de archivos	
Reportes	02/02/2023 12:51	Carpeta de archivos	

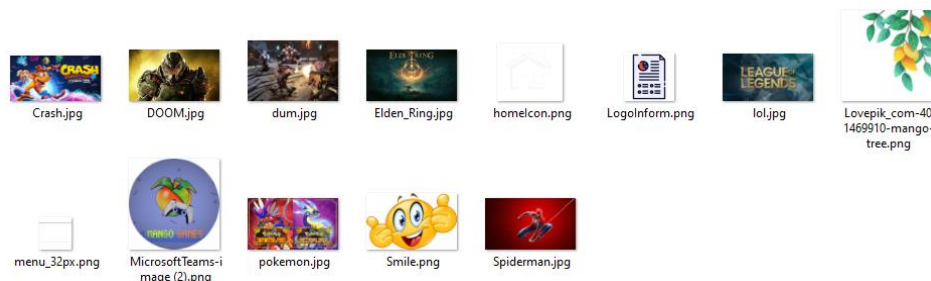
Las partes del proyecto están divididas entre:

-CSV:

Nombre	Fecha de modificación	tipo	tamaño
compras.csv	02/02/2023 12:46	Hoja de cálculo d...	1 KB
juegos.csv	02/02/2023 12:46	Hoja de cálculo d...	2 KB
usuarios.csv	02/02/2023 12:46	Hoja de cálculo d...	1 KB

















Aquí están almacenados los **CSV** para la generación automática de registros en la base de datos cuando el programa se inicie por primera vez en el equipo. Se utiliza para rellenar la tabla **"Compras", "Juegos" y "Usuarios"**. El programa está adaptado para que solo pueda correr en un servidor **MySQL**.

-Images:












Aquí almacenamos todos los **assets** multimedia relacionados con el proyecto. A la hora de registrar un juego tendremos que subir una imagen a la base de datos, es recomendable que esa imagen se guarde aquí si quieres que los cambios persistan.

-Pagina_Home:

 Ayuda.java	02/02/2023 12:46	Archivo de origen ...	2 KB
 Bundle.properties	02/02/2023 12:46	Archivo de origen ...	2 KB
 consultas_sql.java	02/02/2023 12:46	Archivo de origen ...	7 KB
 homeInterface.form	02/02/2023 12:58	Archivo FORM	70 KB
 homeInterface.java	02/02/2023 12:58	Archivo de origen ...	59 KB
 homeMain.java	02/02/2023 12:46	Archivo de origen ...	4 KB
 inicio_sesion.form	02/02/2023 12:46	Archivo FORM	30 KB
 inicio_sesion.java	02/02/2023 12:58	Archivo de origen ...	29 KB
 InterfazJuego.form	02/02/2023 12:59	Archivo FORM	40 KB
 InterfazJuego.java	02/02/2023 12:59	Archivo de origen ...	35 KB
 Juego.java	02/02/2023 12:46	Archivo de origen ...	3 KB
 registro.form	02/02/2023 12:58	Archivo FORM	43 KB
 registro.java	02/02/2023 12:58	Archivo de origen ...	41 KB
 Registro_juegos.form	02/02/2023 12:46	Archivo FORM	71 KB
 Registro_juegos.java	02/02/2023 12:46	Archivo de origen ...	60 KB
 Usuario.java	02/02/2023 12:46	Archivo de origen ...	2 KB

En esta carpeta guardamos el código fuente de la aplicación, la clase **Main** es la de **"homeMain.java"**.

-Reportes:

 Juegos_mas_rentables.jasper	02/02/2023 12:46	Jasper source file	27 KB
 Juegos_mas_rentables.jrxml	02/02/2023 12:46	Jrxml source file	9 KB
 Registro_usuarios_mensuales.jrxml	02/02/2023 12:46	Jrxml source file	8 KB
 Registro_usuarios_mensuales_2.0.jrxml	02/02/2023 12:46	Jrxml source file	9 KB
 Template.jrxml	02/02/2023 12:46	Jrxml source file	4 KB
 Usuarios_mas_rentables.jasper	02/02/2023 12:46	Jasper source file	27 KB
 Usuarios_mas_rentables.jrxml	02/02/2023 12:46	Jrxml source file	9 KB
 usuarios_por_mes.jasper	02/02/2023 12:46	Jasper source file	29 KB
 usuarios_por_mes.jrxml	02/02/2023 12:46	Jrxml source file	10 KB

Esta carpeta almacena los reportes de la aplicación. Están tanto los **.jrxml** como los reportes compilados **.jasper**.

Documentación de las clases de proyecto

Ayuda.java		02/02/2023 12:46	Archivo de origen ...	2 KB
Bundle.properties		02/02/2023 12:46	Archivo de origen ...	2 KB
consultas_sql.java	2	02/02/2023 12:46	Archivo de origen ...	7 KB
homeInterface.form		02/02/2023 12:58	Archivo FORM	70 KB
homeInterface.java	3	02/02/2023 12:58	Archivo de origen ...	59 KB
homeMain.java	1	02/02/2023 12:46	Archivo de origen ...	4 KB
inicio_sesion.form		02/02/2023 12:46	Archivo FORM	30 KB
inicio_sesion.java	4	02/02/2023 12:58	Archivo de origen ...	29 KB
InterfazJuego.form		02/02/2023 12:59	Archivo FORM	40 KB
InterfazJuego.java	5	02/02/2023 12:59	Archivo de origen ...	35 KB
Juego.java		02/02/2023 12:46	Archivo de origen ...	3 KB
registro.form	8	02/02/2023 12:58	Archivo FORM	43 KB
registro.java	6	02/02/2023 12:58	Archivo de origen ...	41 KB
Registro_juegos.form		02/02/2023 12:46	Archivo FORM	71 KB
Registro_juegos.java	7	02/02/2023 12:46	Archivo de origen ...	60 KB
Usuario.java	9	02/02/2023 12:46	Archivo de origen ...	2 KB

1.homeMain

En esta clase se genera en el entorno en el que se va a ejecutar el programa. Aquí se crea la base de datos **"mango_games"** en el caso de que no existiese en el servidor **MySQL** a partir de la clase **"consultas_sql"**.

```
25         LocalDate fecha = LocalDate.now();
26         consultas_sql conexion_db = new consultas_sql("mango_games","root","root");
```

También se generarían las **tablas** de la **base de datos**, con el método **"creacion_tablas()"**, en el caso de que no estuviesen creadas, se crean las tablas: **Usuarios, Juegos y compras**.

```
public static void creacion_tablas(consultas_sql conexion_db) throws SQLException{
    conexion_db.creacion_tabla( "usuarios", "ID int NOT NULL AUTO_INCREMENT,Usuario char(100),Nombre char(100),Apellidos char(200),Email char(200),residencia char(50),clave
    conexion_db.creacion_tabla( "juegos", "ID int NOT NULL AUTO_INCREMENT,Titulo char(100),Descripcion char(200),Precio int,Nota int,Genero char(100),Desarrolladora char(100)
    conexion_db.creacion_tabla( "compras", "ID_usuario int,ID_juego int, Precio_transaccion int ,Fecha_de_compra date,Constraint fk_ID_usuario foreign key (ID_usuario) refe
}
```

Una vez tengamos creadas las tablas, se crearán registros predefinidos a partir de los **CSV** para que las tablas tengan algo de información que manipular. Para esto se comprueba primero si el registro que vamos a meter en las tablas está dentro de la base de datos para no crear duplicados.

```
if(conexion_db.leer_resultset_string(conexion_db.realizar_consulta("select * from usuarios where Usuario='root'"), "Usuario")==null){
    conexion_db.insertar_una_nueva_fila_en_una_tabla("usuarios", "Usuario,Nombre,Apellidos,Email,residencia,clave,administrador,fecha_registro,ultimo_inicio_sesion",
}

ArrayList<String[]> Usuarios = conexion_db.leer_csv(".\\src\\csv\\usuarios.csv");
for(int i = 0; i < Usuarios.size(); i++){
    if(conexion_db.leer_resultset_string(conexion_db.realizar_consulta("select * from usuarios where Usuario='"+ Usuarios.get(i)[0] + "'", "Usuario")==null){
        conexion_db.insertar_una_nueva_fila_en_una_tabla("usuarios", "Usuario,Nombre,Apellidos,Email,residencia,clave,administrador,fecha_registro,ultimo_inicio_sesion",
    }
    else{
        System.out.println("El usuario ya existe");
    }
}

ArrayList<String[]> compras = conexion_db.leer_csv(".\\src\\csv\\compras.csv");
conexion_db.insertar_fichero_csv(compras,"compras","ID_usuario,ID_juego,Precio_transaccion,Fecha_de_compra", false);
homeInterface a = null;
try {
    a = new homeInterface(new Usuario("",false,"0"));
} catch (SQLException ex) {
    Logger.getLogger(homeInterface.class.getName()).log(Level.SEVERE, null, ex);
}
a.setVisible(true);
}
```

2. consultas_sql

Resumen de la clase:

El constructor de esta clase se encarga de crear la conexión a la base de datos **“mango_games”** y tiene como métodos todo lo relacionado con las consultas a la base de datos.

Constructor:

Al constructor hay que pasarle por parámetro: **nombre de la base de datos, usuario y contraseña** para acceder a **“mango_games”**. También comprueba que esté la base de datos creada, se generará en caso contrario.

```
public consultas_sql(String db_,String login_,String password_) throws SQLException {
    // Conexion

    this.connection_ = null;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        this.connection_ = DriverManager.getConnection("jdbc:mysql://localhost:3306/"+db_,login_,password_);
        System.out.println("Conexion a base de datos " + db_ + " correcta.");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println("La base de datos no existe, creando Base de datos...");
        this.connection_ = DriverManager.getConnection("jdbc:mysql://localhost:3306/",login_,password_);
        System.out.println("Puñeta mala");
        Statement creacion_base_datos = this.connection_.createStatement();
        creacion_base_datos.executeUpdate("create database mango_games");
        this.connection_=null;
        this.connection_ = DriverManager.getConnection("jdbc:mysql://localhost:3306/"+db_,login_,password_);
        System.out.println("Conexion a base de datos " + db_ + " correcta.");
        e.printStackTrace();
    }
}
```

Métodos:

Creacion_tabla():

Este método se encarga de crear una tabla en la base de datos, a este método se le debe pasar por parámetro:

- **nombre_tabla** (el nombre de la tabla que vas a crear)
- **atributos** (la cadena de atributos que tendrá la tabla que vayamos a crear).

```
public void creacion_tabla(String nombre_tabla, String atributos) throws SQLException {
    Statement st_;
    st_=this.connection_.createStatement();
    st_.executeUpdate("create table if not exists " + nombre_tabla + "(" + atributos + ")");
}
```

Insertar_una_nueva_fila_en_una_tabla():

Se encarga de introducir los registros que le pasemos por parámetro a la tabla de la base de datos que le designemos. A este método se le debe pasar por parámetro:

- **tabla** (el nombre de la tabla en la que vamos a introducir los datos)
- **atributos** (la cadena de atributos que posee la tabla en la que vamos a introducir los datos)

- **sentencia_atributos**(En este parámetro debemos introducir los valores que tendrá en el registro).

```
public void insertar_una_nueva_fila_en_una_tabla(String tabla, String atributos, String sentencia_atributos) {

    try {
        System.out.println("insert into " + tabla + " (" + atributos + ") values (" + sentencia_atributos + ")");
        Statement st_ = this.connection_.createStatement();
        System.out.println("insert into " + tabla + " (" + atributos + ") values (" + sentencia_atributos + ")");
        st_.executeUpdate("insert into " + tabla + " (" + atributos + ") values (" + sentencia_atributos + ")");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
```

Leer_csv ():

Este método se utiliza para **leer un csv** y guardarlo en un **arraylist de arrays tipo cadena de texto**. Este método recibe por parámetro:

- **fichero_leer**(Le debemos pasar la ruta del archivo que queremos leer)

El método devuelve un objeto tipo **ArrayList<String[]>** con todos los datos del **csv**.

```
public static ArrayList<String[]> leer_csv(String fichero_leer) throws IOException {
    String linea;
    String[] datos;
    ArrayList<String[]> lista = new ArrayList<String[]>();
    try {
        File texto = new File(fichero_leer);
        BufferedReader lectura_fichero = new BufferedReader(
            new InputStreamReader(new FileInputStream(texto), "utf-8"));
        while ((linea = lectura_fichero.readLine()) != null) {
            // Dividir la línea leída por el caractaer ";"
            datos = linea.split(";");
            lista.add(datos);
        }
        return lista;
    } catch (FileNotFoundException fn) {
        ArrayList<String[]> vacio = new ArrayList<>();
        System.out.println("No se encuentra el archivo");
        return vacio;
    } catch (IOException e) {
        ArrayList<String[]> vacio = new ArrayList<>();
        System.out.println("Error de lectura_escritura");
        return vacio;
    }
}
```

Insertar_fichero_csv ():

Esta función se utiliza para introducir el contenido de un **csv** en una de las tablas de la base de datos. Este método se llama normalmente tras la llamada de la función **leer_csv()** ya que hay que pasarle por parámetro un objeto tipo **ArrayList<String[]>**. Se le deben pasar por parámetro:

- **Csv** (Generalmente se deberá introducir el objeto que te devuelve la función **leer_csv()** pero se podrá introducir otro cualquiera)

- **Tabla** (Se le pasa por parámetro el nombre tabla en la que vayamos a introducir los datos)
- **Atributos** (Se deberán introducir los atributos de la tabla correspondiendo a los datos que vamos a introducir)
- **Tiene_cabecera** (Es un valor booleano dependiendo del encabezado de csv, hay algunos csv que tienen un encabezado)

```
public void insertar_fichero_csv(ArrayList<String[]> csv, String tabla, String atributos, boolean tiene_cabecera) throws SQLException {
    if (tiene_cabecera == true) {
        csv.remove(0);
    }

    for (String[] juego : csv) {
        if (leer_resultset_string(realizar_consulta("select * from juegos where Titulo='"+juego[0]+"'", "Titulo") == null){
            insertar_una_nueva_fila_en_una_tabla(tabla, atributos, parsear_atributos(juego));
        }
    }

    System.out.println("DATOS DE " + tabla + " INSERTADOS");
}
```

Realizar_consulta ():

Este método devuelve un Resultset en base a una consulta SQL que le pasamos por parámetro:

- **Query** (En este parámetro se le deberá de pasar una **consulta SQL** para que pueda devolver un **Resultset** en base a la **query** pasada por parámetro)

```
public ResultSet realizar_consulta(String query) throws SQLException {

    Statement _st = this.connection_.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    System.out.println(query);
    ResultSet _rs = _st.executeQuery(query);

    return _rs;
}
```

Leer_resultset_string/int ():

Estas funciones se utilizan para leer los **Resultset** que normalmente nos devuelve el método **realizar_consulta()** para que nos devuelva una cadena de texto o un carácter numérico. **Este método solo se utiliza con Resultsets que solo contengan un valor.**

- **_rs** (Aquí se introduce el **Resultset** que quieras leer, el método no funcionará correctamente si el **Resultset** tiene más de un dato)
- **Atributo** (Se le pasa por parámetro una cadena de texto con el nombre de la columna que quieres que devuelva)

```

public String leer_resultset_string(ResultSet _rs, String atributo) throws SQLException {
    _rs.beforeFirst();
    String valorpedido = null;
    while (_rs.next()) {

        valorpedido = _rs.getString(atributo);

    }
    return valorpedido;
}

public static int leer_resultset_int(ResultSet _rs, String atributo) throws SQLException {
    _rs.beforeFirst();
    int valorpedido = 0;
    while (_rs.next()) {

        valorpedido = _rs.getInt(atributo);

    }
    return valorpedido;
}

```

IsParsable():

Este método comprueba si una cadena de texto se puede convertir a **int** o **no**. Para saber esto, la función te devuelve un **valor booleano**. Se le debe de pasar por parámetro:

- **Input** (Se debe escribir la cadena de texto que quieras comprobar)

```

public static boolean isParsable(String input) {
    try {
        Integer.parseInt(input);
        return true;
    } catch (final NumberFormatException e) {
        return false;
    }
}

```

Parsear_cadena():

Esta función se ayuda del método **isParsable()** para colocar **comillas dobles** a la cadena de texto que le pases por parámetro. Este método se utiliza normalmente para formatear los atributos a lo hora forma una **query** de manera automática con un **csv**.

- **Cadena**(Aquí se introduce la cadena de texto que queremos formatear)

```

public String parsear_cadena(String cadena) {
    return (isParsable(cadena)) ? cadena : "\"" + cadena + "\"";
}

```

Parsear_atributos():

Este método se utiliza para formatear un conjunto de atributos en un array de cadenas de texto y prepararlas para introducirlas en la query que vayamos a lanzar. Este

método ya devuelve la cadena de texto formateada. Este método utiliza el método `parsear_cadena()` para realizar su trabajo.

- **Atributos** (Se le proporcionará un array con los datos que deben formatearse)

```
public String parsear_atributos(String[] atributos) {  
  
    String sentencia_atributos = "";  
  
    // Crea una sentencia compatible con MySQL  
    for (int j = 0; j < atributos.length; j++) {  
  
        atributos[j]=this.parsear_cadena(atributos[j]);  
  
        sentencia_atributos += atributos[j];  
        if (j != atributos.length - 1) {  
            sentencia_atributos += ",";  
        }  
  
    }  
    return sentencia_atributos;  
}
```

3.homeInterface



Resumen de la clase:

Esta es la pantalla principal de nuestro proyecto, cuando iniciamos la aplicación dirigirá automáticamente a esta pantalla. Aquí los usuarios podrán buscar los artículos que los usuarios quieran buscar, tanto por nombre como por género.

Constructor:

En el constructor se comprueba si se ha iniciado sesión en la aplicación con el método **comprobarInicioS()**. También se recogen los datos de los juegos almacenados en la base de datos para mostrárselos al usuario cuando abra la ventana.

```
//Se utiliza para terminar conseguir el numero de columnas del result set
ResultSetMetaData rsmd = resultSet.getMetaData();
int totalColumnas = rsmd.getColumnCount();

while (resultSet.next()) {
    //Pilla todos los atributos de cada registro
    for(int i=1; totalColumnas>=i; i++){
        datosJuego.add(resultSet.getString(i));
    }
    //Crea el onjeto juego y lo introduce en el array de objetos juegos
    juegos.add(new Juego(datosJuego));
    datosJuego.clear();
}
}
```

Al constructor hay que pasarle por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo **“Usuario”**, esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

Métodos:

ComprobarInicioS():

Este método se encarga de comprobar si se ha iniciado sesión y también si la ha iniciado un administrador. A este método se le debe pasar por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo **“Usuario”**, esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

```
public void comprobarInicioS(Usuario usuario){
    if(usuario.getNombre_usuario().equals("")==false){
        esAdmin(usuario.getAdmin());
        ajustes.setVisible(true);
        this.UserLabel.setText("Bienvenido "+this.usuario_i.getNombre_usuario());
        UserLabel.setVisible(true);
        loginButton.setVisible(false);
        registerButton.setVisible(false);
    }else{
        ajustes.setVisible(false);
        UserLabel.setVisible(false);
        loginButton.setVisible(true);
        registerButton.setVisible(true);
    }
}
```

esAdmin():

Esta función se encarga de comprobar si el usuario que ha iniciado sesión es administrador o no. A este método se le debe pasar por parámetro:

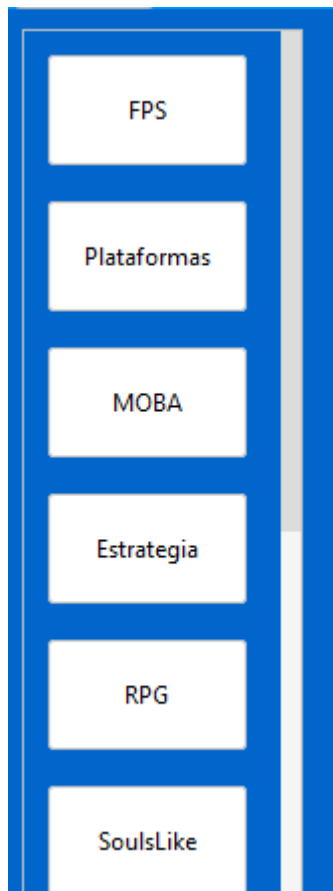
- **Admin** (Esta variable está almacenada en un atributo dentro del objeto **Usuario**, este atributo demuestra si el usuario que ha iniciado sesión es administrador o no)

```
public void esAdmin(boolean admin){  
    if(admin==true){  
        botonAdmin.setVisible(true);  
    }else{  
        botonAdmin.setVisible(false);  
    }  
}
```

busquedaCategorias():

Este método se encarga de buscar los juegos por categorías dentro de los datos que hemos recogido de la base de datos al iniciar la aplicación, en el objeto **Juego** hay un atributo que utilizamos para identificar cuál es su género. A este método se le debe pasar por parámetro:

- **Categoria**(Le mandaremos por parámetro la categoría que queremos que busque, este parámetro se le proporcionará a partir de los botones que tendremos en el lateral izquierdo de la pantalla)



```

public void busquedaCategorias(String categoria){
    this.search.clear();

    JLabel[] labelJuego = {
        G1,
        G2,
        G3,
        G4,
        G5,
        G6
    };
    JButton[] imagenJuego = {
        B1,
        B2,
        B3,
        B4,
        B5,
        B6
    };
    for(int j=0;labelJuego.length>j;j++){
        labelJuego[j].setText("");
    }

    for(int i = 0; juegos.size()>i;i++){
        if(categoria.equalsIgnoreCase(juegos.get(i).getGenero())){
            search.add(juegos.get(i));
        }else{
            for(int j=0;labelJuego.length>j;j++){
                System.out.println("entra");
                labelJuego[j].getParent().setVisible(false);
                sinResult.setVisible(true);
            }
        }
    }
    for(int j=0;search.size()>j;j++){
        labelJuego[j].setText(search.get(j).getTitulo());
        Image img=getToolkit().getImage(search.get(j).getImagen());
        img=img.getScaledInstance(B1.getWidth(), B1.getHeight(), Image.SCALE_DEFAULT);
        imagenJuego[j].setIcon(new ImageIcon(img));
    }
    for(int j=0;labelJuego.length>j;j++){
        if(labelJuego[j].getText().equalsIgnoreCase("")){
            labelJuego[j].getParent().setVisible(false);
        }else{
            labelJuego[j].getParent().setVisible(true);
            sinResult.setVisible(false);
        }
    }
}
}

```

4.inicio_sesion

Resumen de la clase:

Esta es la pantalla que utilizará el usuario para iniciar sesión. Al igual que en las demás pantallas, se comprobará si algún usuario ha iniciado sesión con anterioridad.

Constructor:

En el constructor se comprueba si se ha iniciado sesión en la aplicación con el método **comprobarInicioS()**. Al constructor hay que pasarle por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo “**Usuario**”, esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

Métodos:

Boton_Inicio_SesionActionPerformed():

Esta función se utiliza cuando el usuario pulsa el botón de **iniciar sesión**. En el caso de que se haya iniciado sesión correctamente redirigirá al usuario a la página de **homeInterface**, pero en el caso contrario, no te dejará iniciar sesión con las credenciales incorrectas y te mostrará un mensaje de error.

Una vez se hayan introducido las credenciales, estas se compararán con la base de datos y se mostrará si las credenciales son correctas. En este proceso también se comprobará si el usuario que intenta iniciar sesión es administrador.

```
private void boton_inicio_sesionActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_boton_inicio_sesionActionPerformed
    // TODO add your handling code here:
    boolean admin;
    String nombre_usuario;
    String id;
    char [] clave_array = clave_inicio.getPassword();
    String pass = "";

    for (int j = 0; j < clave_array.length; j++) {
        pass+=clave_array[j];
    }

    try {
        ResultSet usuario = conexion_db.realizar_consulta("select * from usuarios where (usuario='"+conexion_db.parsear_cadena(usuario_correo.getText())+" or email='"+conexion_db.parsear_cadena(usuario_email.getText())+"' and clave='"+pass+"')");
        if(conexion_db.leer_resultset_string(usuario, "Usuario")!=null){

            if(conexion_db.leer_resultset_string(usuario, "administrador").equalsIgnoreCase("1")){
                admin = true;
            }else{
                admin = false;
            }

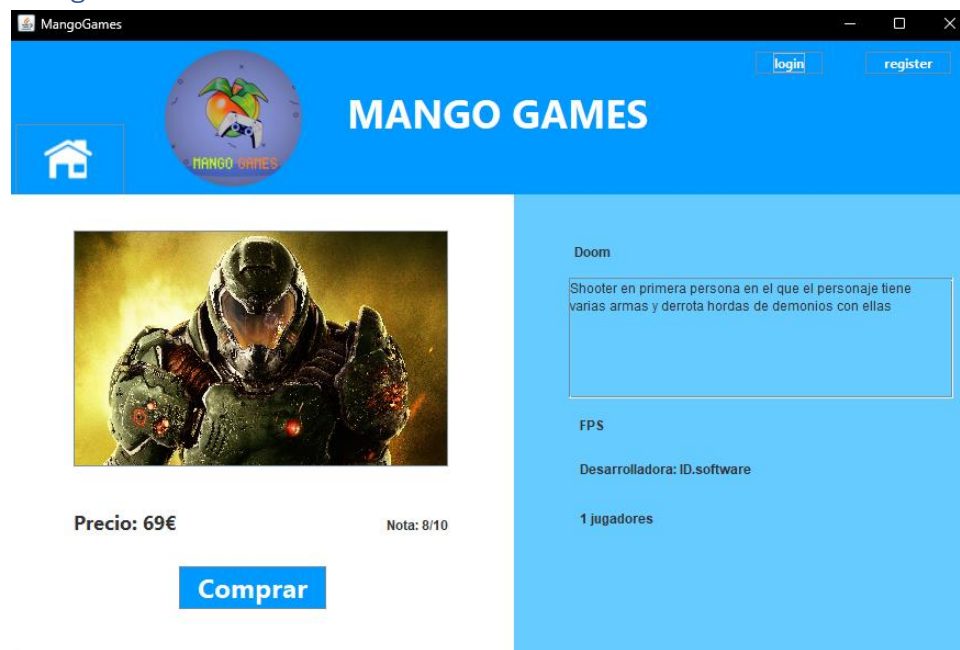
            nombre_usuario=conexion_db.leer_resultset_string(usuario, "Usuario");
            id=conexion_db.leer_resultset_string(usuario, "ID");
            this.usuario_i = new Usuario(nombre_usuario,admin,id);
            this.UserLabel.setText("Bienvenido "+this.usuario_i.getNombre_usuario());
            this.UserLabel.setVisible(true);
            this.ajustes.setVisible(true);
            this.botonRegistro.setVisible(false);
            homeInterface a = null;

            try {
                a = new homeInterface(usuario_i);
            } catch (SQLException ex) {
                Logger.getLogger(homeInterface.class.getName()).log(Level.SEVERE, null, ex);
            }

            a.setVisible(true);
            this.setVisible(false);
            LocalDate fecha = LocalDate.now();
            conexion_db.actualizar_resultset_string(usuario, "ultimo_inicio_sesion", conexion_db.formatear_fecha(fecha));

        }else{
            String nombre = conexion_db.leer_resultset_string(usuario, "Usuario");
            String correo = conexion_db.leer_resultset_string(usuario, "email");
            System.out.println(nombre);
            System.out.println(correo);
            System.out.println(conexion_db.leer_resultset_string(usuario, "clave"));
            if(nombre == null || correo == null || conexion_db.leer_resultset_string(usuario, "clave") == null){
                JOptionPane.showMessageDialog(rootPane, "ERROR: LOS DATOS INTRODUCIDOS SON ERRONEOS");
            }
        }
    } catch (SQLException ex) {
        Exceptions.printStackTrace(ex);
    }
}
```


5. Interfaz Juego



Resumen de la clase:

A esta pantalla se puede acceder desde la **pantalla principal** tras pinchar en alguno de los **juegos** mostrados en ellas. Aquí podrás ver los datos sobre el juego elegido (precio, título, género, etc.) Aquí podrás comprar los juegos, pero para ello tendrás que iniciar sesión. Si no estas **logueado** en la página, esta te mostrará una pantalla en el que podrás iniciar sesión o registrarte.

Constructor:

En el constructor se construye la interfaz a través de los parámetros que le pasamos desde la página principal (Título, Descripción, precio, etc.)

```

public InterfazJuego(Juego juego, Usuario usuario) throws SQLException {
    this.usuario_i=usuario;
    initComponents();
    ajustes2.setVisible(false);
    miniMenu2.setVisible(false);
    comprobarInicioS(usuario_i);
    this.Confirmacion_compra.setVisible(false);
    this.Confirmacion_compra.setEnabled(false);
    this.panel_excepcion_compra.setVisible(false);
    this.panel_excepcion_compra.setEnabled(false);
    consultas_sql conexion_db = new consultas_sql("mango_games","root","root");
    this.conexion_db = new consultas_sql("mango_games","root","root");
    this.juego = juego;
    this.LabelDescripcion.setEditable(false);
    this.LabelDescripcion.setLineWrap(true);
    this.LabelDescripcion.setWrapStyleWord(true);

    this.ImagenJuego.setIcon((new ImageIcon(convertir_imagen())));
    this.LabelTitulo.setText(this.juego.getTitulo());
    this.LabelDescripcion.setText(this.juego.getDescripcion());
    this.LabelDesarrolladora.setText("Desarrolladora: " +this.juego.getDesarrolladora());
    this.LabelPrecio.setText("Precio: " + String.valueOf(this.juego.getPrecio()) + "€");
    this.LabelNota.setText("Nota: " +this.juego.getNota() + "/10");
    this.LabelNjugadores.setText(this.juego.getNumero_jugadores() + " jugadores");
    this.LabelGenero.setText(this.juego.getGenero());
}

```

Los parámetros que se utilizan son:

- **Juego** (Este objeto guarda las características del juego que queremos ver los detalles o comprar)
- **Usuario** (Esta instancia contiene la información del usuario que ha iniciado sesión)

Métodos:

Convertir_Imagen():

Esta función extrae la imagen del objeto **Juego** y lo convierte en un objeto tipo **Image** y lo devuelve.

```

public Image convertir_imagen(){
    System.out.println(this.juego.getImagen());
    Image img=getToolkit().getImage(this.juego.getImagen());
    img=img.getScaledInstance(ImagenJuego.getWidth(), ImagenJuego.getHeight(), Image.SCALE_DEFAULT);
    return img;
}

```

ComprobarInicioS():

Este método se encarga de comprobar si se ha iniciado sesión y también si la ha iniciado un administrador. A este método se le debe pasar por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo **“Usuario”**, esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

```

public void comprobarInicioS(Usuario usuario){
    if(usuario.getNombre_usuario().equals("")==false){
        esAdmin(usuario_i.getAdmin());
        ajustes2.setVisible(true);
        this.UserLabel.setText("Bienvenido "+this.usuario_i.getNombre_usuario());
        UserLabel.setVisible(true);
        Boton_login.setVisible(false);
        Boton_registro.setVisible(false);
    }else{
        ajustes2.setVisible(false);
        UserLabel.setVisible(false);
        Boton_login.setVisible(true);
        Boton_registro.setVisible(true);
    }
}
}

```

esAdmin():

Esta función se encarga de comprobar si el usuario que ha iniciado sesión es administrador o no. A este método se le debe pasar por parámetro:

- **Admin** (Esta variable está almacenada en un atributo dentro del objeto **Usuario**, este atributo demuestra si el usuario que ha iniciado sesión es administrador o no)

```

public void esAdmin(boolean admin){
    if(admin==true){
        botonAdmin2.setVisible(true);
    }else{
        botonAdmin2.setVisible(false);
    }
}
}

```

Boton_comprarActionPerformed():

Este método te comprueba si estás logeado en la página para comprar el artículo. En caso contrario, te mostrará la ventana para iniciar sesión o registrarte. Si está todo correcto, se mandará un **INSERT** para registrar la nueva compra en la base de datos.

```

private void Boton_comprarActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_Boton_comprarActionPerformed
    // TODO add your handling code here:
    this.Confirmacion_compra.setVisible(true); //Lo he puesto a true para que se active al pulsar de momento, al ejecutar he puesto que sea invisible(VONLEE)
    LocalDate fecha = LocalDate.now();

    if(this.usuario_i.getNombre_usuario()!=""){
        conexion_db.insertar_una_nueva_fila_en_una_tabla("compras", "id_usuario,id_juego,precio_transaccion,fecha_de_compra", conexion_db.parsear_cadena(usuario_i.getId()));
        this.Confirmacion_compra.setVisible(true);
        this.Confirmacion_compra.setEnabled(true);
    }else{
        //NECESITAS INICIAR SESION O REGISTRARTE
        this.panel_excepcion_compra.setVisible(true);
        this.panel_excepcion_compra.setEnabled(true);
    }
}
}

```

6.registro



Resumen de la clase:

El usuario se dirigirá a esta pantalla si quiere registrar una nueva cuenta en nuestra base de datos. El usuario deberá proporcionar los datos de manera adecuada para poder registrar la cuenta correctamente, la aplicación no le dejará avanzar hasta que no tenga escrito los campos correctamente.

Constructor:

```
public registro(Usuario usuario_tranferido) throws SQLException {
    this.usuario_i=usuario_tranferido;
    initComponents();
    comprobarInicio(usuario_i);
    miniMenu.setVisible(false);
    ajustes.setVisible(false);
    UserLabel.setVisible(false);
    ValidationGroup Campo_Validator_usuario = Validator_usuario.getValidationGroup();
    ValidationGroup Campo_Validator_nombre = Validator_nombre.getValidationGroup();
    ValidationGroup Campo_Validator_apellido = Validator_apellidos.getValidationGroup();
    ValidationGroup Campo_Validator_correo = Validator_correo.getValidationGroup();
    ValidationGroup Campo_Validator_clave = Validator_clave.getValidationGroup();
    Campo_Validator_usuario.add(usuario,StringValidators.NO_WHITESPACE,StringValidators.REQUIRE_NON_EMPTY_STRING);
    Campo_Validator_nombre.add(nombre,StringValidators.REQUIRE_NON_EMPTY_STRING);
    Campo_Validator_apellido.add(apellidos,StringValidators.REQUIRE_NON_EMPTY_STRING);
    Campo_Validator_correo.add(correo_electronico,StringValidators.REQUIRE_NON_EMPTY_STRING,StringValidators.EMAIL_ADDRESS);

    Validator_usuario.addChangeListener(newChangeListener() {
        @Override
        public void stateChanged(ChangeEvent e) {
            System.out.println(terminos.condiciones.isContentAreaFilled());
            if(Validator_usuario.getProblem() == null && Validator_nombre.getProblem() == null && Validator_apellidos.getProblem() == null && Validator_correo.getProblem() == null)
                registro.setEnabled(true);
            }else{
                registro.setEnabled(false);
            }
        }
    });
    this.setIconImage(new ImageIcon(getClass().getResource("/images/MicrosoftTeams-image (2).png")).getImage());
    this.conexion_db = new consultas_sql("mango_games","root","root");
}
```

En el constructor se crean los objetos tipo **“ValidationGroup”** para revisar la introducción de los datos en el formulario por parte del usuario y no haya errores a la hora de registrar la nueva cuenta en la base de datos. También revisa que todos los campos estén correctos con la desactivación del botón **“Registrar”** hasta que todo esté correctamente.

Los parámetros que se utilizan son:

- **Usuario_tranferido** (Esta instancia contiene la información del usuario que ha iniciado sesión)

Métodos:

borrar_Campos():

Este método se utiliza para borrar todos los campos del formulario que han sido rellenados. Normalmente se llama a este método cuando se pulsa el botón de registro, una vez se hayan verificado que los campos estén correctos.

```
private void borrar_campos() {  
    this.usuario.setText("");  
    this.nombre.setText("");  
    this.apellidos.setText("");  
    this.correo_electronico.setText("");  
    this.clave.setText("");  
    this.terminos_condiciones.setSelected(false);  
}
```

ComprobarInicioS():

Este método se encarga de comprobar si se ha iniciado sesión y también si la ha iniciado un administrador. A este método se le debe pasar por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo "**Usuario**", esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

```
public void comprobarInicioS(Usuario usuario){  
    if(usuario.getNombre_usuario().equals("")==false){  
        esAdmin(usuario_i.getAdmin());  
        ajustes2.setVisible(true);  
        this.UserLabel.setText("Bienvenido "+this.usuario_i.getNombre_usuario());  
        UserLabel.setVisible(true);  
        Boton_login.setVisible(false);  
        Boton_registro.setVisible(false);  
    }else{  
        ajustes2.setVisible(false);  
        UserLabel.setVisible(false);  
        Boton_login.setVisible(true);  
        Boton_registro.setVisible(true);  
    }  
}
```

esAdmin():

Esta función se encarga de comprobar si el usuario que ha iniciado sesión es administrador o no. A este método se le debe pasar por parámetro:

- **Admin** (Esta variable está almacenada en un atributo dentro del objeto **Usuario**, este atributo demuestra si el usuario que ha iniciado sesión es administrador o no)

```

public void esAdmin(boolean admin){
    if(admin==true){
        botonAdmin2.setVisible(true);
    }else{
        botonAdmin2.setVisible(false);
    }
}

```

RegistroActionPerformed():

En esta función se formatea el campo “contraseña” para convertirlo en una cadena de texto y subirlo a la base de datos. También se encarga de revisar que no haya ningún usuario que se llame del mismo modo que el usuario registrado. Una vez estén todos los campos correctamente, se podrá insertar en la base de datos el nuevo juego

```

private void registroActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_registroActionPerformed
    try {
        // TODO add your handling code here:
        char [] clave_array = clave.getPassword();
        String pass = "";

        for (int j = 0; j < clave_array.length; j++) {
            pass+=clave_array[j];
        }

        String [] datos_usuario = {usuario.getText(),nombre.getText(),apellidos.getText(),correo_electronico.getText(),residencia.getSelectedIndex().toString(),pass};
        String usuario= ""+datos_usuario[0]+"";
        usuario=conexion_db.leer_resultset_string(conexion_db.realizar_consulta("select * from usuarios where usuario="+usuario, "usuario");
        if(usuario==null){
            LocalDate fecha = LocalDate.now();
            conexion_db.insertar_una_nueva_fila_en_una_tabla("usuarios", "Usuario,Nombre,Apellidos,Email,residencia,clave,administrador,fecha_registro,ultimo_inicio_sesion");
            borrar_campos();
            JOptionPane.showMessageDialog(rootPane, "USUARIO REGISTRADO");
        }else{
            JOptionPane.showMessageDialog(rootPane, "ERROR: USUARIO YA REGISTRADO");
        }
    } catch (SQLException ex) {
        Exceptions.printStackTrace(ex);
    }
}
}

```

7.Registro_juegos

Resumen de la clase:

A esta ventana solo podrán acceder los usuarios que tengan las credenciales de **administrador**. En esta ventana podremos registrar un nuevo juego en nuestra base de datos, todos los campos deben estar bien escritos para poder insertar el juego en la base de datos. También podremos crear nuevos informes con los botones que proporcionamos, algunos de estos informes requerirán de unos parámetros para poder generarse.

Constructor:

```
public Registro_juegos(Usuario usuario) throws SQLException {
    this.usuario_i=usuario;
    initComponents();
    comprobarInicio(usuario_i);
    this.conexion_db = new consultas_sql("mango_games","root","root");
    ResultSet juegos = conexion_db.realizar_consulta("select * from juegos");
    System.out.println(juegos);
    this.setIconImage(new ImageIcon(getClass().getResource("/images/MicrosoftTeams-image (2).png")).getImage());

    ValidationGroup Campo_Validador_titulo = Validador_titulo.getValidationGroup();
    ValidationGroup Campo_Validador_descripcion = Validador_descripcion.getValidationGroup();
    ValidationGroup Campo_Validador_precio = Validador_precio.getValidationGroup();
    Campo_Validador_titulo.add(titulo,StringValidators.REQUIRE_NON_EMPTY_STRING);
    Campo_Validador_descripcion.add(descripcion,StringValidators.REQUIRE_NON_EMPTY_STRING);
    Campo_Validador_precio.add(precio,StringValidators.REQUIRE_NON_EMPTY_STRING,StringValidators.REQUIRE_VALID_NUMBER);

    Validador_titulo.addChangeListener(new ChangeListener() {
        @Override
        public void stateChanged(ChangeEvent e) {

            if(Validador_titulo.getProblem() == null && Validador_descripcion.getProblem() == null && Validador_precio.getProblem() == null){
                registro_juego.setEnabled(true);
            }else{
                registro_juego.setEnabled(false);
            }
        }
    });
}
```

En el constructor se crean los objetos tipo **"ValidationGroup"** para revisar la introducción de los datos en el formulario por parte del usuario y no haya errores a la hora de registrar la nueva cuenta en la base de datos. También revisa que todos los campos estén correctos con la desactivación del botón **"Registrar"** hasta que todo esté correctamente.

Los parámetros que se utilizan son:

- **Usuario_tranferido** (Esta instancia contiene la información del usuario que ha iniciado sesión)

Métodos:

borrar_Campos():

Este método se utiliza para borrar todos los campos del formulario que han sido rellenados. Normalmente se llama a este método cuando se pulsa el botón de registro, una vez se hayan verificado que los campos estén correctos.

```
private void borrar_campos() {
    this.usuario.setText("");
    this.nombre.setText("");
    this.apellidos.setText("");
    this.correo_electronico.setText("");
    this.clave.setText("");
    this.terminos_condiciones.setSelected(false);
}
```

ComprobarInicioS():

Este método se encarga de comprobar si se ha iniciado sesión y también si la ha iniciado un administrador. A este método se le debe pasar por parámetro:

- **Usuario** (Le debemos pasar por parámetro un objeto tipo “**Usuario**”, esto se utiliza para pasar los datos del usuario entre las distintas ventanas de la aplicación)

```
public void comprobarInicioS(Usuario usuario){
    if(usuario.getNombre_usuario().equals("")==false){
        esAdmin(usuario_i.getAdmin());
        ajustes2.setVisible(true);
        this.UserLabel.setText("Bienvenido "+this.usuario_i.getNombre_usuario());
        UserLabel.setVisible(true);
        Boton_login.setVisible(false);
        Boton_registro.setVisible(false);
    }else{
        ajustes2.setVisible(false);
        UserLabel.setVisible(false);
        Boton_login.setVisible(true);
        Boton_registro.setVisible(true);
    }
}
```

esAdmin():

Esta función se encarga de comprobar si el usuario que ha iniciado sesión es administrador o no. A este método se le debe pasar por parámetro:

- **Admin** (Esta variable está almacenada en un atributo dentro del objeto **Usuario**, este atributo demuestra si el usuario que ha iniciado sesión es administrador o no)

```
public void esAdmin(boolean admin){
    if(admin==true){
        botonAdmin2.setVisible(true);
    }else{
        botonAdmin2.setVisible(false);
    }
}
```

CreacionInformes():

Este método se ocupa de la generación de los informes; crea una ruta absoluta a partir de una ruta relativa mediante el método **.getAbsolutePath()** dado de netbeans no acepta rutas relativas. Después, rellena el reporte con los datos de la base de datos (JasperPrint jp) y lo muestra por pantalla (*JasperViewer.viewReport(jp)*)

- El atributo *necesitaPar* es dado desde la llamada al método y sirve para saber si el reporte necesita parámetros, lo que en la ventana de la aplicación muestra una ventana con *textInput* para introducir estos parámetros (*menuDatos.setVisible(true)*)


```
public void creacionInformes(Boolean necesitaPar){
    if(necesitaPar){
        menuDatos.setVisible(true);
    }else{
        try{
            //JasperReport jr= JasperCompileManager.compileReport(this.rutacarpeta.getAbsolutePath());
            System.out.println(this.rutacarpeta.getAbsolutePath());
            JasperPrint jp = JasperFillManager.fillReport(this.rutacarpeta.getAbsolutePath(),null, conexion_db.getConnection_() );
            JasperViewer.viewReport(jp);

        }catch(Exception ex){
            ex.printStackTrace();
        }
        menuDatos.setVisible(false);
    }
}
```

8.Juego

Resumen de la clase:

La clase juego consiste en un constructor de objetos que simulan ser los juegos de una tienda online, para más tarde usarse a la hora de mostrar los datos necesarios de los mismos a petición del usuario en las pantallas `home_interface` e `interfaz_juego`

Constructor:

El constructor se compone de las distintas características que tiene un juego real; un ID, un nombre y una descripción del mismo, un precio por el que comprarlo, una nota media resultante de valoraciones de otros usuarios, además de unos datos que para detallar la modalidad de juego además de la empresa encargada del desarrollo.

Estos atributos se pasarán desde un `arrayList` llamado ***datos*** que recoge los datos de un archivo `.csv` que contiene todos los juegos de nuestra base de datos:

```
//Constructor con el mismo nombre de la clase
public Juego(ArrayList <String> datos){
    this.ID = datos.get(0);
    this.Título = datos.get(1);
    this.Descripcion = datos.get(2);
    this.Precio = datos.get(3);
    this.Nota = datos.get(4);
    this.Genero = datos.get(5);
    this.Desarrolladora = datos.get(6);
    this.Numero_jugadores = datos.get(7);
    this.imagen = datos.get(8);
}
```

Por supuesto, además del constructor esta clase posee métodos ***get*** y ***set*** para cada uno de los atributos de la misma.

Métodos:

Esta clase no tiene ningún método a excepción de los ya comentados

9.Usuario

Resumen de la clase:

La clase usuario, al igual que la de juego, consiste en un constructor de objetos, los cuales consisten en usuarios de la aplicación, mediante el cual se guardarán en la base de datos registros de compras, inicios de sesión...

Constructor:

El constructor de la clase usuario se compone de los atributos ***admin*** (indica si el usuario es administrador o no), ***nombre_usuario*** (el nombre que un usuario introduce al registrarse) e ***ID*** (número de identificación):

```
public Usuario(String nombre_usuario,boolean admin,String id) throws SQLException {  
    this.admin = admin;  
    this.nombre_usuario = nombre_usuario;  
    this.id = id;  
}
```

Y por supuesto, los métodos **get** y **set** correspondientes de los atributos

Métodos:

Esta clase no tiene ningún método a excepción de los ya comentados