CICLO: SEGUNDO DAM

MÓDULO DE PROGRAMACIÓN DE SERVICIOS Y PROCESOS

Tarea Nº 05

Alumno:
Ramón Martínez Nieto
43188174-Q

Contenido

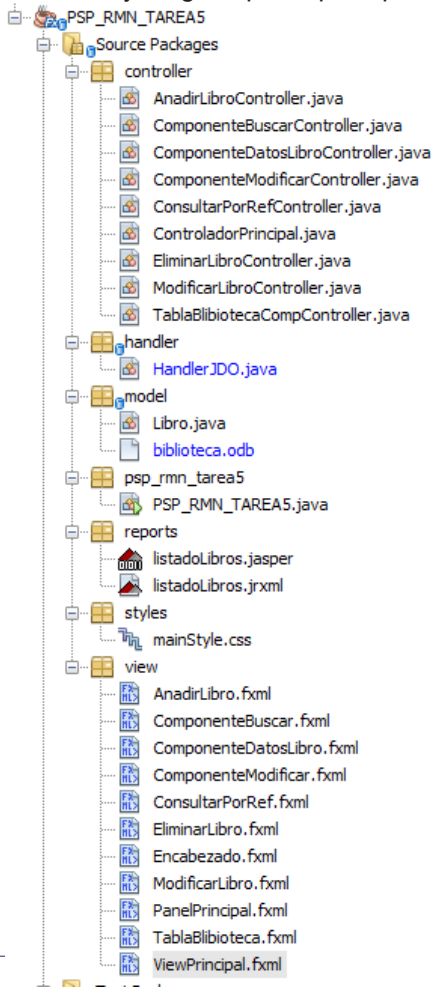
1. Documentos que se adjuntan a este informe.

A continuación se detallan los documentos que componen la presente entrega de la tarea:

1. Informe de elaboración de la tarea.
2. Carpeta proyecto cliente: PSP_RMN_TAREA5

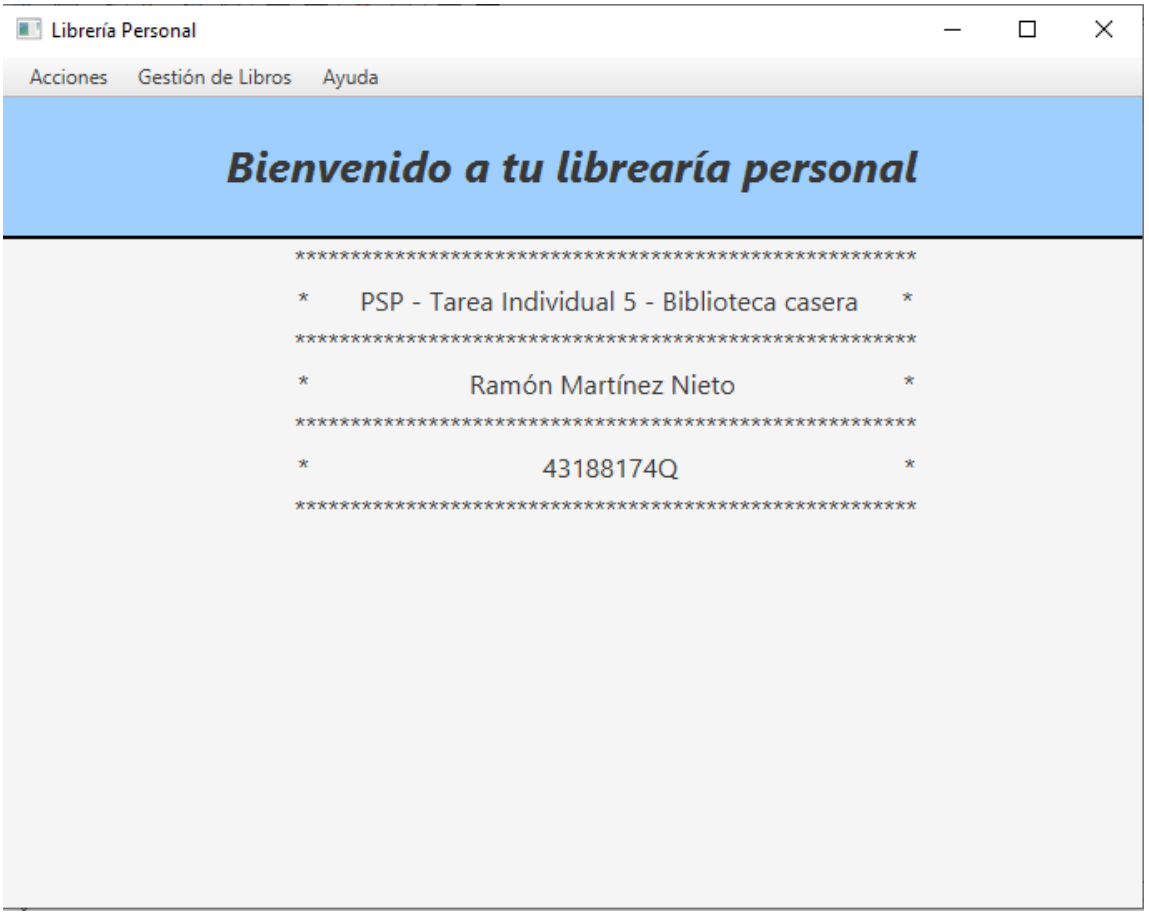
2. Introducción.

Para practicar conceptos aprendidos en otras asignaturas como Desarrollo de Interfaces y aprender a integrar los conceptos en una misma aplicación, he desarrollado para la “Biblioteca Personal” una interfaz creada con JavaFXML, también he incluido la posibilidad de generar un informe de los libros almacenados en la biblioteca. Comentar que he creado diferentes archivos .fxml para crear la aplicación a base de componentes re-utilizables, estos archivos .fxml son las parte vista del modelo MVC, patrón que he seguido para la tarea, el paquete controller contiene los controlladores (Controlador), el paquete model contiene la clase Libro y el archivo con los datos (Modelo). Además hay otros paquetes como reports que contiene un archivo para la creación de un informe con todos los libros, el paquete handler dónde está la clase HandlerJDO (principal objetivo de esta tarea), el paquete styles que contiene el archivo .css para manejar el estilo de la interfaz y el paquete psp_rmn_tarea5 que contiene el método main y carga el panel principal.

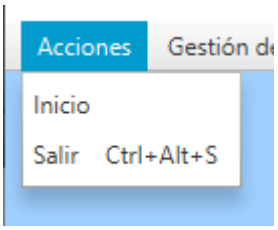


Como el objetivo de la tarea es aprender a trabajar con la persistencia de datos mediante el uso de JDO, he elaborado el informe de la tarea centrándome en las clases y objetos para su cumplimiento, obviando una extensa explicación de la creación de toda la interfaz, a excepción de esta introducción para saber como funciona y de que paneles se compone poder dar cumplimiento también a la parte explícita del menú.

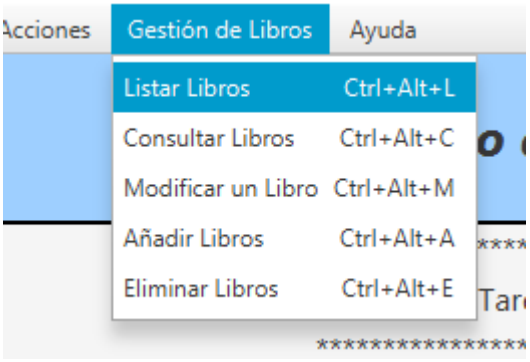
Interfaz de la aplicación.



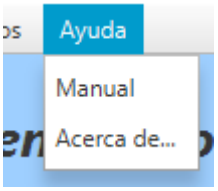
En la pantalla inicial de la interfaz podemos ver como se muestran mis datos, requerimiento para poder entregar la tarea. También vemos claramente el menú que tiene la interfaz, está separado por “Acciones”, “Gestion de Libros” y “Ayuda. Desde Acciones podemos ir al panel principal o salir.



Desde el menú “Gestión de Libros” podemos realizar las diferentes tareas que nos solicitan la interfaz como; consultar los datos, listar los datos, modificar los datos almacenados y eliminar datos. Además he añadido la opción “Añadir Libro”.

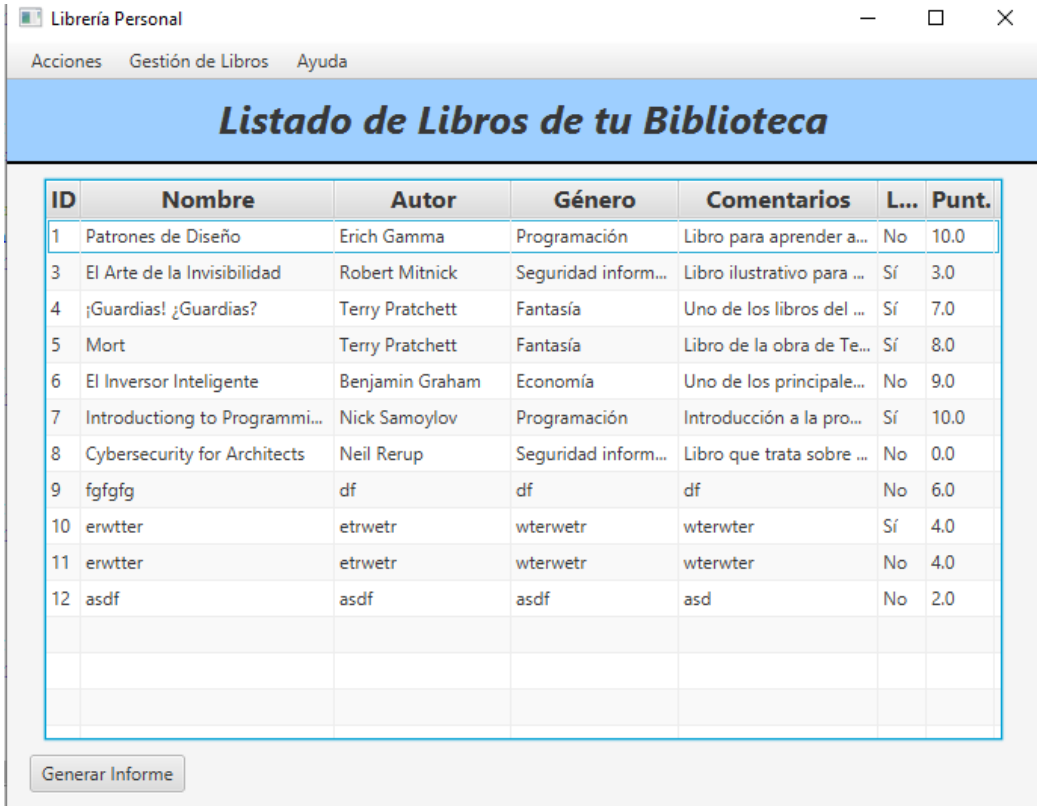


En el menú ayuda tenemos dos opciones; manual y Acerca de… El Manual abrirá este informe y "Acerca de… " mostrará mis datos personales.



Como he comentado este menú está creado en JavaFXML, y éste está creado en el archivo "ViewPrincipal.fxml" que además contiene un panel para ir cargando los diferentes componentes.

Centrándome en el apartado de la gestión de libros con el uso de JDO, si pulsamos sobre la opción "Listar Libros" tenemos la siguiente interfaz, dónde tendremos un listado de todos los libros almacenados y un botón para generar un informe de los libros.



ID	Nombre	Autor	Género	Comentarios	L...	Punt.
1	Patrones de Diseño	Erich Gamma	Programación	Libro para aprender a...	No	10.0
3	El Arte de la Invisibilidad	Robert Mitnick	Seguridad inform...	Libro ilustrativo para ...	Sí	3.0
4	¡Guardias! ¿Guardias?	Terry Pratchett	Fantasía	Uno de los libros del ...	Sí	7.0
5	Mort	Terry Pratchett	Fantasía	Libro de la obra de Te...	Sí	8.0
6	El Inversor Inteligente	Benjamin Graham	Economía	Uno de los principale...	No	9.0
7	Introductiong to Programmi...	Nick Samoylov	Programación	Introducción a la pro...	Sí	10.0
8	Cybersecurity for Architects	Neil Rerup	Seguridad inform...	Libro que trata sobre ...	No	0.0
9	fgfgfg	df	df	df	No	6.0
10	erwtter	etrwetr	wterwetr	wterwter	Sí	4.0
11	erwtter	etrwetr	wterwetr	wterwter	No	4.0
12	asdf	asdf	asdf	asd	No	2.0

En el apartado “Consultar Libros” podemos consultar cualquier libro a través de su ID.

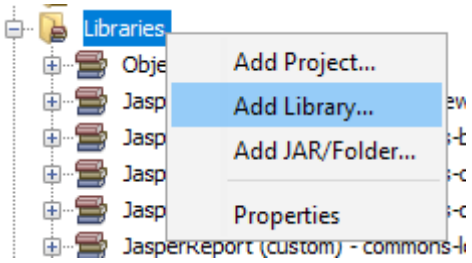
En el apartado modificar libros podremos modificar un libro que previamente deberemos de buscar.

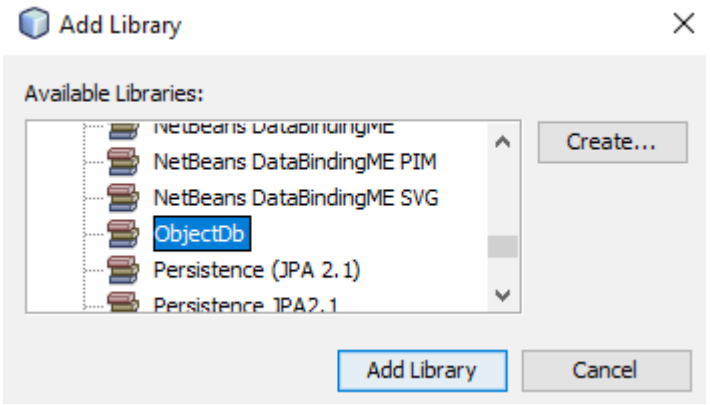
En el apartado “Eliminar Libro” tenemos la opción de eliminar un libro que hayamos buscado previamente.

Y para finalizar la opción añadir libro nos permitirá agregar nuevos libros.

3. Importar librerías necesarias.

Para el desarrollo de esta tarea es necesario importar la librería ObjectDb que nos brindará de todo lo necesario para poder trabajar con persistencia de datos mediante JDO. Para la importación de la librería pulsamos el botón derecho sobre “Libraries” y le damos a la opción “Add Library”. Se abrirá una ventana donde podemos escoger de diferentes librerías, si ya habíamos agregado la librería de ObjectDb nos aparecerá, de lo contrarío podemos crearla a través del botón “Create...” y seleccionar la librería en cuestión. En mi caso ya la tengo creada y solo tengo que seleccionarla y pulsar sobre “Add Library”.





Nota: También uso la librería JasperReport para la creación del informe.

4. Clase Libro.java.

La clase libro representa un libro de nuestra librería. Para el correcto funcionamiento con la persistencia de datos con JDO la clase tiene que ir precedida de la etiqueta @Entity e implementar la interfaz Serializable.

El primer atributo que declaro es el id de tipo int, pero con las etiquetas @Id y @GeneratedValue delante para identificar el atributo como el id de la clase y que se genere automáticamente. Posteriormente tengo las variables especificadas en la tarea. Hay una variable extra llamada “leido2” de tipo boolean, esta variable la cree con el fin de crear correctamente el informe y el ObservableList para la tabla de JavaFX y mostrar los valores “Sí” y “No” en ver de “true” o “false” cuando se muestren las opciones de si se ha leído o no el libro.

```
14  public class Libro implements Serializable {
15
16      //id generado automáticamente y único
17      @Id
18      @GeneratedValue
19      private int id;
20
21      String nombre, autor, genero, comentarios;
22      boolean leido; // Si o no
23      /** Este dato lo uso para el ObservableList, con ello pu
26      boolean leido2;
27
28      double puntuacion;  // 0 - 10
29
```

A continuación tenemos d os constructores, uno que recibe todos los parámetros (no recibe el valor leído dos veces, se utiliza el mismo) y otro vacío.

```
33  /** Constructor completo ...10 lines */
43  public Libro(String nombre, String autor, String genero, String comentarios, boolean leido, double puntuacion) {
44      this.nombre = nombre;
45      this.autor = autor;
46      this.genero = genero;
47      this.comentarios = comentarios;
48      this.leido = leido;
49      this.puntuacion = puntuacion;
50
51      this.leido2 = leido;
52  }
53
54  /** Constructor vacio ...3 lines */
57  public Libro() {
58  }
```

La continuación de la clase son los setter y getter normales de cualquier clase tipo POJO, a excepción del atributo de tipo boolean isLeido2 que tiene un trato especial, para recuperar un String diciendo “Sí” o “No” como expliqué anteriormente.

```
110     public String isLeido2() {
            String leido = "No";
112
113         if(this.leido){
114             leido = "Sí";
115         }
116
117         return leido;
118     }
119
```

5. Clase HadnlerJDO.

La clase HandlerJDO.java es la clase principal de la que se basa esta tarea, decidí crear esta clase para manejar los objetos persistentes desde una sola clase, así pues las clases controladoras de la interfaz solo tienen que crear un objeto de esta clase y llamar al método que corresponda cada vez.

La clase comienza con la declaración de una constante indicando la ruta al archivo dónde se harán persistentes los objetos, también declaro dos objetos, uno de tipo EntityManagerFactory y otro del tipo EntityManajer, cabe destacar que estos métodos no los instancio en este momento.

```
11  public class HandlerJDO {
12
13      //Ruta para guardar los datos JDO en el paquete model
14      private final String rutaBiblioteca = "./src/model/biblioteca.odb";
15
16      EntityManagerFactory entityMF;
17      EntityManager entityM;
18
```

Los primeros métodos que quiero comentar son los métodos openEntity() y closeEntity(). Entendiendo la importancia que tiene el abrir y cerrar los objetos del tipo Entity para que el archivo no se quede abierto por error, cree dos métodos para este fin, así también evito repetir el mismo código. El método openEntity() primero instancia el objeto EntityManager a través de la clase Persistence y el método createEntittyManagerFactory al que le paso la ruta de la biblitoeca. El método closeEntity() llama a los métodos close() para cerrar ambos objetos y así no tener ningún flujo abierto hacia el archivo.

```
229     /** Método para instanciar el EntitityManagerFactory y el EntitityManager ...3 li
232     private void openEntity() {
233         entityMF = Persistence.createEntityManagerFactory(rutaBiblioteca);
234         entityM = entityMF.createEntityManager();
235     }
236
237     /** Método para cerrar el EntityManagerFactory y EntityManager ...3 lines */
240     private void closeEntity() {
241         entityM.close();
242         entityMF.close();
243     }
```

Como para evitar que el usuario cometa algún error y la aplicación falle, una parte del control de excepciones la hice lanzando objetos Alert al usuario para notificarle del error y que así pueda corregirlo. Este método construye un Alert recibiendo por parámetros el tipo de Alert, el titulo y el header, además está sobrecargado para poder recibir también el content. Los métodos terminan mostrando el mensaje al usuario.

```
266     /** Método para mostrar un Alert ...6 lines */
272     private void creacionAlert(Alert.AlertType tipo, String titulo, String header) {
273         Alert a = new Alert(tipo);
274         a.setTitle(titulo);
275         a.setHeaderText(header);
276         a.show();
277     }
278
279     /** Método sobrecargado para mostrar un alert con un Content ...7 lines */
286     private void creacionAlert(Alert.AlertType tipo, String titulo, String header,String content){
287         Alert a = new Alert(tipo);
288         a.setTitle(titulo);
289         a.setHeaderText(header);
290         a.setContentText(content);
291         a.show();
292     }
```

El método isEmpty(String str) lo cree para controlar que a la hora de introducir datos al menos esté el Nombre del libro, se podría utilizar para comprobar cualquier campo deseado.

```
294     /** Comprobar si un campo está vacio ...6 lines */
300     private boolean isEmpty(String str) {
301         return str.equals("");
302     }
```

El método isNumeric(String puntuacion) lo cree para controlar que los datos de un TextField sean de tipo double, para poder manejar si el usuario se equivocó. Otra opción era crear un TextField personalizado que solo permita introducir caracteres numéricos al usuario pero ya se estaba complicando demasiado la interfaz para esta tarea.

```
304     /** Comprobar si un String es número de tipo double ...6 lines */
310     private boolean isNumeric(String puntuacion) {
311         try {
312             Double.parseDouble(puntuacion);
313             return true;
314         } catch (NumberFormatException e) {
315             return false;
316         }
317     }
```

El método comprobaciones(String puntuacion, String nombre) lo cree para comprobar ambos campos, realmente solo lo utilizo dos veces, pero para no hacer demasiado largos otros métodos decidí separarlo. Este método recibe dos String, realiza la comprobación del String puntuación tanto si es numérico y si está en el rango entre y 0 y 10, posteriormente comprueba en otro else if si el campo nombre está vacío, de ser todo correcto no entrará en ningún if saltando el else y poniendo la variable de tipo boolean en true confirmando que ambos String son correctos. Podemos observar que si entra en alguna condición la variable de tipo boolean se pondrá a false y se creará un objeto del tipo Alert a través del método comentado anteriormente.

```
245     /** Método para comprobar los campos puntuación y nombre ...6 lines */
251     private boolean comprobaciones(String puntuacion, String nombre) {
            boolean correcto = false;
253         //Compruebo los datos de la puntuación y el nombre, si todo está bien almaceno el libro
254
255         if (!isNumeric(puntuacion)) {
256             creacionAlert(Alert.AlertType.ERROR, "Error con la puntuación", "La puntuación tiene quer de tipo entero o double.");
257             correcto = false;
258
259         } else if (Double.parseDouble(puntuacion) < 0 || Double.parseDouble(puntuacion) > 10) {
260             creacionAlert(Alert.AlertType.ERROR,"Error con la puntuación","La puntuación tiene que estar entre 0 y 10");
261             correcto = false;
262         } else if (isEmpty(nombre)) {
263             creacionAlert(Alert.AlertType.ERROR, "Error con el nombre", "El Nombre del libro no puede estar en blanco.");
264             correcto = false;
265         } else {
266             correcto = true;
267         }
268
269         return correcto;
270     }
```

Ahora ya procedo a explicar los métodos que controlar el archivo con el uso de JDO. El primer método es consultarBiblioteca() este método devuelve una colección de libros “List<Libro>”. Se instancia el EntitityManagerFactory y el EntityManager con el método openEntity() explicado, se crea un objeto del tipo TypeQuery<Libro> y a través del EntityManager creamos una consulta con el método createQuery. Almacenamos el resultado en una colección del tipo List<Libro> usando el método getResultList() del objeto TypeQuery<Libro>, cerramos ambos Entity (método closeEntitty()) y devolvemos está lista con el return.

```
19      /** Método para consultar la lista de libros ...4 lines */
23      public List<Libro> consultarBiblioteca() {
24
25          openEntity();
26
27          //Creo un TypeQuery para crear una consulta y consultar todos los libros
            TypedQuery<Libro> consulta = (TypedQuery<Libro>) entityM.createQuery("SELECT libro FROM Libro libro");
29
30          //Almaceno los resultados en un List
31          List<Libro> listaLibros = consulta.getResultList();
32
33          //Cierro el entity
34          closeEntity();
35
36          return listaLibros;
37      }
```

El método consultarLibro(int id) que devuelve un objeto del tipo Libro, nos servirá para consultar un solo libro a través del id recibido por parámetro (este id a su vez se extrae de un TextField que ha introducido el usuario a través de la interfaz). Se llama al método openEntity() (para instanciar los Entity), se vuelve a crear un objeto del tipo TypeQuery<Libro> y crear la consulta con createQuery(), y almacenado todos los libros en una colección tipo List<Libro>. Ahora creo un objeto de tipo Libro instanciado a null, con un bucle foreach itero sobre la lista de libros comprobando el id de cada libro con el id recibido por parámetro, cuando hay una coincidencia se almacena el libro el el Libro, se cierra los Entity con closeEntity() y por si algún casual el archivo se hubiera corrompido, antes de devolver el Libro con return compruebo si el Libro sigue a null, ya que a pesar de controlar las excepciones al ser un archivo puede corromperse o borrarse directamente, así que con un if compruebo si el libro sigue a null, de seguir a null informo a través de un Alert y devuelvo null, si no entrara en este condicional entonces es que hay un libro encontrada y devuelvo dicho libro.

```java
/** Método para consultar un solo libro ...6 lines */
public Libro consultarLibro(int id) {

    openEntity();
    //Creo un TypeQuery para crear una consulta y consultar todos los libros
    TypedQuery<Libro> consulta = (TypedQuery<Libro>) entityM.createQuery("SELECT libro FROM Libro libro");
    List<Libro> libros = consulta.getResultList();

    Libro libroEncontrado = null;
    for (Libro l : libros) {
        //Busco el libro que me interesa a través del número del TextField
        if (l.getId() == id) {
            libroEncontrado = l;
        }
    }
    //Cierro los Entity para que no permanezcan abiertos
    closeEntity();

    //Compruebo si se encontró el libro, de lo contrario muestro un alert
    if (libroEncontrado == null) {
        creacionAlert(Alert.AlertType.INFORMATION, "Error al buscar el libro", "El libro que intentas buscar no está almacenado.");

        return null;
    } else {
        return libroEncontrado;
    }
}
```

Uno de los últimos métodos es el de eliminarLibro(int id), a través del identificador recibido pro parámetro podemos eliminar un libro concreto. Para ello se llama al método openEntity(), se crea de nuevo el objeto TypeQuery<Libro> y se crea la consulta de los libros que haya en el archivo con el método createQuery y se pasa a la colección con el método getResultList(). Para la eliminación del libro es necesario iniciar una transacción, para ello utilizo los métodos getTransaction() y begin() del objeto EntityManager, posteriormente con un bucle for itero sobre la colección comprobando el id de cada libro, en el momento que encuentre una coincidencia puedo eliminar ese Libro usando el método remove(Libro) de objeto EntityManager. El método commit() de getTransaction() del EntittyManager lo pongo en un bloque try así si saltara una excepción a la hora de eliminar puedo controlarla y crear un Alert que informe al usuario, además de informar al usuario se hará un rollback() de la operación de eliminación para revertir los cambios usando el método rollback(). Para finalizar cierro con el método closeEntitty() que llama a los métodos close() del EntityManager y EntitityManagerFactory.

```java
/** Método para eliminar un libro a través del id ...4 lines */
public void eliminarLibro(int id) {
    openEntity();

    TypedQuery<Libro> consulta = (TypedQuery<Libro>) entityM.createQuery("SELECT libro FROM Libro libro");
    List<Libro> libros = consulta.getResultList();

    //inicio una transacción
    entityM.getTransaction().begin();

    for (Libro l : libros) {
        //Busco el libro que me interesa a través del número del TextField
        if (l.getId() == id) {
            entityM.remove(l);
        }
    }

    try {
        //hago los cambios persistentes
        entityM.getTransaction().commit();
    } catch (Exception e) {
        creacionAlert(Alert.AlertType.ERROR, "Error al elimnar.", "El libro no ha podido ser elimnado.");
        entityM.getTransaction().rollback();
    }
    //cierro el Entity
    closeEntity();
}
```

El método que viene a continuación es sin duda el más extenso, este método servirá para modificar un Libro. El usuario a través de la interfaz buscará el libro que quiere modificar, una vez encontrado se habilitarán los TextField con los datos del libro encontrado, ahora el usuario puede editar dichos datos y darle al botón modificar.

El método modificarLibro() recibe por parámetro todos los datos de los TextFields, nótese que el TextField correspondiente al id está en gris, esto es por que no se puede modificar este campo pero se utilizará para encontrar el libro a modificar que también se recibe por parámetro en el método modificarLibro(), no escogí el campo que se utiliza para indicar el libro a buscar por que ese campo lo podría modificar el usuario sin querer en cualquier momento y lanzaría una excepción o haría los datos del archivo inconsistentes modificando un libro con otro id. Una vez entra en el método se comprueba si la puntuación está vacía, de estar vacía se pone por defecto a 0.0 como tipo String. Después se llama a openEntitty() y se crean los objetos TypedQuery y List<Libro> como en los anteriores métodos, ahora se procede a iterar sobre la lista de libros y buscar la coincidencia a través del id recibido por parámetro. Cuando se encuentra el libro a modificar en un if se comprueba con el método comprobaciones() los campos puntuación y nombre, de ser todo correcto se inicia la transacción con getTransaction().begin(), se establecen los valores recibidos por parámetro, haciendo un cast al tipo Double de la puntuación ya que del TextField se recibe un String, hago el commit() en un bloque try para controlar una posible excepción, de hacerse el commit entonces creo un String con los datos del libro y muestro un Alert al usuario informando que el libro a sido modificado y los datos que se han guardado. Si saltara el catch muestro un alert diferente al usuario y hago un rollback(). Para finalizar cierro los objetos Entitty a través del método closeEntity().

```java
/** Método para modificar un libro ...11 lines */
public void modificarLibro(int id, String nombre, String autor, String genero, String comentario, boolean leido, String puntuacion) {
    if (isEmpty(puntuacion)) {
        puntuacion = "0.0";
    }
    openEntity();

    TypedQuery<Libro> consulta = (TypedQuery<Libro>) entityM.createQuery("SELECT libro FROM Libro libro");
    List<Libro> libros = consulta.getResultList();

    for (Libro l : libros) {
        //Busco el libro que me interesa a través del número del TextField
        if (l.getId() == id) {

            if (comprobaciones(puntuacion, nombre)) {
                //Ahora realizo la modificación
                entityM.getTransaction().begin();

                l.setAutor(autor);
                l.setNombre(nombre);
                l.setGenero(genero);
                l.setComentarios(comentario);
                l.setLeido(leido);
                l.setPuntuacion(Double.parseDouble(puntuacion));

                try {

                    //hago los cambios persistentes
                    entityM.getTransaction().commit();

                    String setContentText = "Datos: \n"
                            + "Nombre:\t\t" + l.getNombre() + "\n"
                            + "Autor:\t\t" + l.getAutor() + "\n"
                            + "Genero:\t\t" + l.getGenero() + "\n"
                            + "Comentarios:\t\t" + l.getComentarios() + "\n"
                            + "Leido:\t\t" + l.isLeido() + "\n"
                            + "Puntuación:\t\t" + l.getPuntuacion();

                    creacionAlert(Alert.AlertType.INFORMATION, "Modificado", "Libro " + l.getNombre() + " modificado con éxito.", setContentText);
                } catch (Exception e) {
                    creacionAlert(Alert.AlertType.WARNING, "Error al modificar el libro", "No se ha podido modificar el libro.");
                    entityM.getTransaction().rollback();
                }
            }
        }
    }

    //cierro el Entity
    closeEntity();
}
```

El método sirve para almacenar un nuevo libro, ya que es una librería personal tuve a bien crear la opción de agregar nuevos libros.

El método almacenarLibro() recibe por parámetro los datos del Libro, esta vez no es necesario el id, como antes compruebo si la puntuación esta vacía para ponerla a 0.0 si fuera así y poder hacer un parse. Abro el flujo con el método openEntitty(), compruebo los datos puntuación y nombre con el método comprobaciones(), inicio una transacción con getTransaction().begin() y hago persistentes los cambios con el método persist() del objeto EntittyManager, al método persist le paso un nuevo libro que contiene los datos que recibe el método. Para finalizar en un bloque try hago el commit() de los cambios y muestro un alert informativo al usuario, como el método anterior si salta el catch entonces hago un rollback() y muestro un Alert de Error, el método finaliza cerrando los fljos con el método closeEntitty().

```java
/** Método para almacenar un nuevo libro ...10 lines */
public void almacenarLibro(String nombre, String autor, String genero, String comentario, boolean leido, String puntuacion) {
    if (isEmpty(puntuacion)) {
        puntuacion = "0.0";
    }

    //Inicio el Entity
    openEntity();
    if (comprobaciones(puntuacion, nombre)) {
        //Inicio una transacción
        entityM.getTransaction().begin();
        //almaceno datos de muestra
        entityM.persist(new Libro(nombre, autor, genero, comentario, leido, Double.parseDouble(puntuacion)));

        //Hago persistentes los datos
        try {
            entityM.getTransaction().commit();

            String setContentText = "Datos: \n"
                    + "Nombre:\t\t" + nombre + "\n"
                    + "Autor:\t\t" + autor + "\n"
                    + "Genero:\t\t" + genero + "\n"
                    + "Comentarios:\t\t" + comentario + "\n"
                    + "Leido:\t\t" + leido + "\n"
                    + "Puntuación:\t\t" + puntuacion;

            creacionAlert(Alert.AlertType.INFORMATION, "Libro agregado", "Libro " + nombre + " agregado con éxito.", setContentText);
        } catch (Exception e) {
            creacionAlert(Alert.AlertType.ERROR, "Error al elimnar.", "El libro no ha podido ser elimnado.");
            entityM.getTransaction().rollback();
        }
    }
    //Cierro el entity
    closeEntity();

}
```

El último método se llama almacenarLibroSimple(), este método tiene la única función de cargar libros de ejemplo, lo hice así por que los libros los instancio manualmente y puedo asegurarme que no hay errores en los datos. Instancia los Entity, inicia la transacción hace persistente una instancia de un objeto Libro con los parámetros recibidos a través de los parámetros del método, hace el commit y cierra los Entitty con el método closeEntitty().

```java
/** Método para almacenar un nuevo libro de forma simple ...10 lines */
public void almacenarLibroSimple(String nombre, String autor, String genero, String comentario, boolean leido, String puntuacion) {

    //Inicio el Entity
    openEntity();

    //Inicio una transacción
    entityM.getTransaction().begin();
    //almaceno datos de muestra
    entityM.persist(new Libro(nombre, autor, genero, comentario, leido, Double.parseDouble(puntuacion)));
    //Hago persistentes los datos
    entityM.getTransaction().commit();

    closeEntity();
}
```

6. Otras clases y métodos.

El ControladorPrincipal.java es el controlador de la interfaz PanelPrincipal, se úbican los métodos necesarios para el funcionamiento del menú (para cambiar de panel). Básicamente hay un método llamado cambiarPanel() que recibe la ruta del panel a cargar y lo carga utilizando las clases y métodos de JavaFX para este fin.

```
86      /** Método para cambiar de panel ...5 lines */
91      private void cambiarPanel(String panelSwitch) {
92          panelPrincipal.getChildren().clear();
93          URL url = getClass().getResource(panelSwitch);
94
95          try {
96              Node nodo = FXMLLoader.load(url);
97
98              panelPrincipal.getChildren().add(nodo);
99              AnchorPane.setTopAnchor(nodo, 0.0);
00              AnchorPane.setLeftAnchor(nodo, 0.0);
01              AnchorPane.setRightAnchor(nodo, 0.0);
02              AnchorPane.setBottomAnchor(nodo, 0.0);
03
04          } catch (IOException ex) {
05              System.err.println("Error al cambiar de panel");
06          }
07      }
```

Podemos ver que los métodos a los que llama el menú son muy simples y solo llaman al método anterior con la ruta especificada.

```
20  public class ControladorPrincipal implements Initializable {
21
22      //rutas a las diferentes vistas componentes
23      private final String TABLA_BIBLIOTECA = "/view/TablaBlibioteca.fxml";
24      private final String CONSULTAR_REF = "/view/ConsultarPorRef.fxml";
25      private final String EILIMNAR_LIBRO = "/view/EliminarLibro.fxml";
26      private final String MODIFICAR_LIBRO = "/view/ModificarLibro.fxml";
27      private final String PANEL_PRINCIPAL = "/view/PanelPrincipal.fxml";
28      private final String ANADIR_LIBRO = "/view/AnadirLibro.fxml";
29
30      //Panel Principal
31      @FXML
32      AnchorPane panelPrincipal;
33
34      @FXML
35      private void verTablaBiblioteca(ActionEvent event) { cambiarPanel(TABLA_BIBLIOTECA);    }
36
37      @FXML
38      private void panelInicio(ActionEvent event) { cambiarPanel(PANEL_PRINCIPAL);    }
39
40      @FXML
41      private void consultarLibro(ActionEvent event) { cambiarPanel(CONSULTAR_REF);    }
42
43      @FXML
44      private void modificarLibro(ActionEvent event) { cambiarPanel(MODIFICAR_LIBRO);    }
45
46      @FXML
47      private void elminarLibro(ActionEvent event) { cambiarPanel(EILIMNAR_LIBRO);    }
48
49      @FXML
50      private void anadirLibro(ActionEvent event) { cambiarPanel(ANADIR_LIBRO);    }
51
```

Lo más importante de esta clase es que es la encargada de cargar los libros de muestra. Para ello tiene un método que instancia un objeto del tipo HandlerJDO (clase explicada anteriormente), a través de este objeto se llama al método almacenarLibroSimple y se le pasan los datos de cada método.

```
/** Método para cargar libros de muestra ...5 lines */
private void cargarLibrosMuestra() {
    //Clase HandlerJDO para manejar el archivo
    HandlerJDO jdo = new HandlerJDO();

    jdo.almacenarLibroSimple("Patrones de Diseño", "Erich Gamma", "Programación", "Libro para aprender a usar los patrones de diseño", false, "10.0");
    jdo.almacenarLibroSimple("Mastering JavaFX10", "Sergey Ginev", "Programación", "Libro para aprender a programar con JvaFX", false, "6.0");
    jdo.almacenarLibroSimple("El Arte de la Invisibilidad", "Robert Mitnick", "Seguridad informática", "Libro ilustrativo para aplicar seguridad informátic
    jdo.almacenarLibroSimple("¡Guardias! ¿Guardias?", "Terry Pratchett", "Fantasia", "Uno de los libros del mundo creado por Terry Pratchett", true, "7.0")
    jdo.almacenarLibroSimple("Mort", "Terry Pratchett", "Fantasia", "Libro de la obra de Terry Pratchett, mirando el mundo como la muerte", true, "8.0");
    jdo.almacenarLibroSimple("El Inversor Inteligente", "Benjamin Graham", "Economía", "Uno de los principales libros sobre la economía y como invertir", f
    jdo.almacenarLibroSimple("Introductiong to Programming", "Nick Samoylov", "Programación", "Introducción a la programación con Java", true, "10.0");
    jdo.almacenarLibroSimple("Cybersecurity for Architects", "Neil Rerup", "Seguridad informática", "Libro que trata sobre la ciberseguridad", false, "0.0"

}

@Override
```

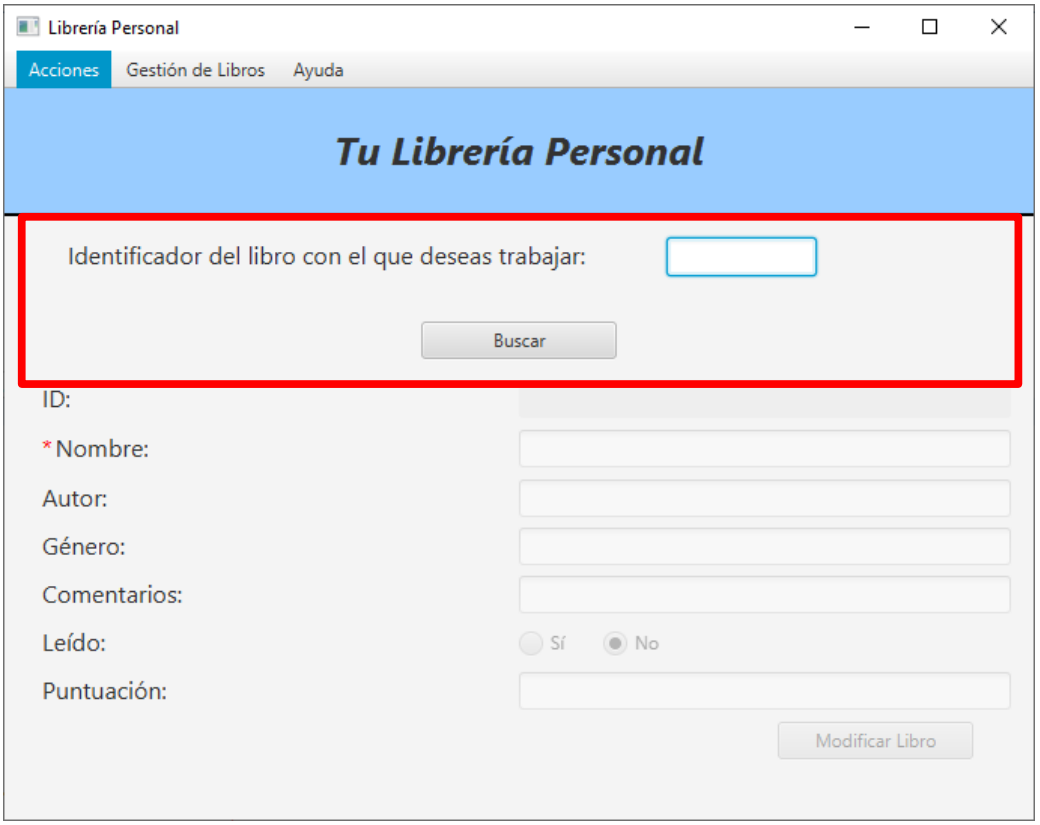
Este método será llamado en el método sobreescrito que usamos en JavaFX al arrancar la aplicación, primero cargará el panel principal, posteriormente comprobará si ya existe un archivo llamado “biblioteca.odb” usando un objeto File, si el objeto no existe entonces llamo al método anterior, que creará y cargará los libros de muestra, si el archivo ya existe entonces este método no se ejecuta.

```
139     @Override
        public void initialize(URL url, ResourceBundle rb) {
141         //Cargo el panel principal al entrar en la aplicación
142         cambiarPanel(PANEL_PRINCIPAL);
143
144         File f = new File("./src/model/biblioteca.odb");
145         //Si el archivo no existe entonces lo creo y cargo los libros de muestra
146         if (!f.exists()) {
147             cargarLibrosMuestra();
148         }
149     }
150 }
```

Otro método que me parece importante es el método buscarLibro(), este método es el encargado de llamar al método consultarLibro() alojado en el HandlerJDO, así pues cuando el usuario introduce un id en el componente para buscar un libro este es recogido y pasado a consultarLibro del Handler, con los datos devueltos entonces establecemos los valor se los Labels dónde se muestra la información.

```
41
42      /** Método para buscar el Libro y establecer los datos ...4 lines */
46      public void buscarLibro(int id) {
47          HandlerJDO jdo = new HandlerJDO();
48          limpiarDatos();
49
50          if (id != -1) {
51              Libro l = jdo.consultarLibro(id);
52
53              if (l != null) {
54                  setLblId(String.valueOf(l.getId()));
55                  setLblNombre(l.getNombre());
56                  setLblAutor(l.getAutor());
57                  setLblGenero(l.getGenero());
58                  setLblComentarios(l.getComentarios());
59                  setLblLeido(l.isLeido());
60                  setLblPuntuacion(String.valueOf(l.getPuntuacion()));
61              }
62          }
63      }
64
```

Como la aplicación está basada en componentes, el método anterior es una funcionalidad del componente “buscar” que podemos encontrar en los paneles “Consultar Libro”, “Modificar Libro” y “Eliminar Libro”. Así he aprovechado el código lo máximo posible.



Lo marcado en un recuadro rojo es el componente buscar.

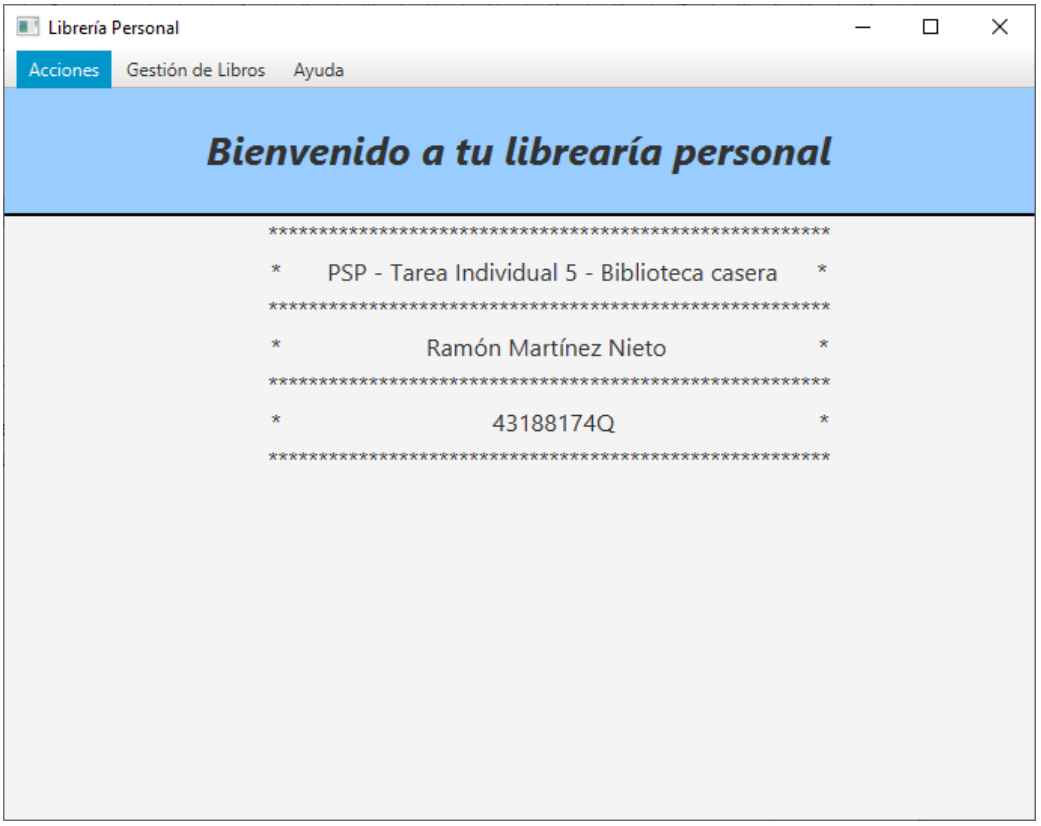
Por ejemplo, la clase ConsultarPorRefController, que controla el panel dónde se cargan los datos de un libro que especifiquemos llamará al método buscar descrito anteriormente y luego establecerá los campos en el componente que corresponda. Al haber creado la interfaz por componentes, a la hora de crear nuevos paneles resulta muy sencillo y solo se tienen que escribir un par de líneas de código para completar la funcionalidad.

```
9   public class ConsultarPorRefController implements Initializable {
10
11      @FXML
12      ComponenteDatosLibroController componenteDatosController;
13
14      @FXML
15      ComponenteBuscarController componenteBuscarController;
16
17      /** Método para el botón buscar por referencia ...4 lines */
21      @FXML
22      public void buscar(ActionEvent event) {
23          componenteDatosController.buscarLibro(componenteBuscarController.getFieldId());
24      }
25
26      @Override
28      public void initialize(URL url, ResourceBundle rb) {
29
29      }
30  }
```

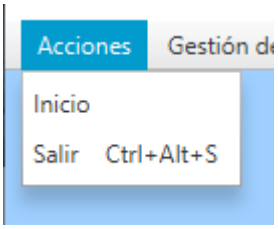
Para no extender más la explicación de la tarea y haber abordado con profundidad los puntos que se requieren no voy a explicar el resto de clases.

7. Ejecución de la aplicación.

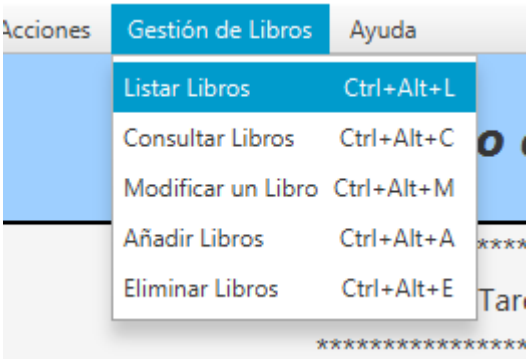
Inicio de la aplicación:



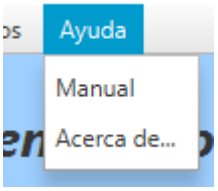
Menú Acciones:



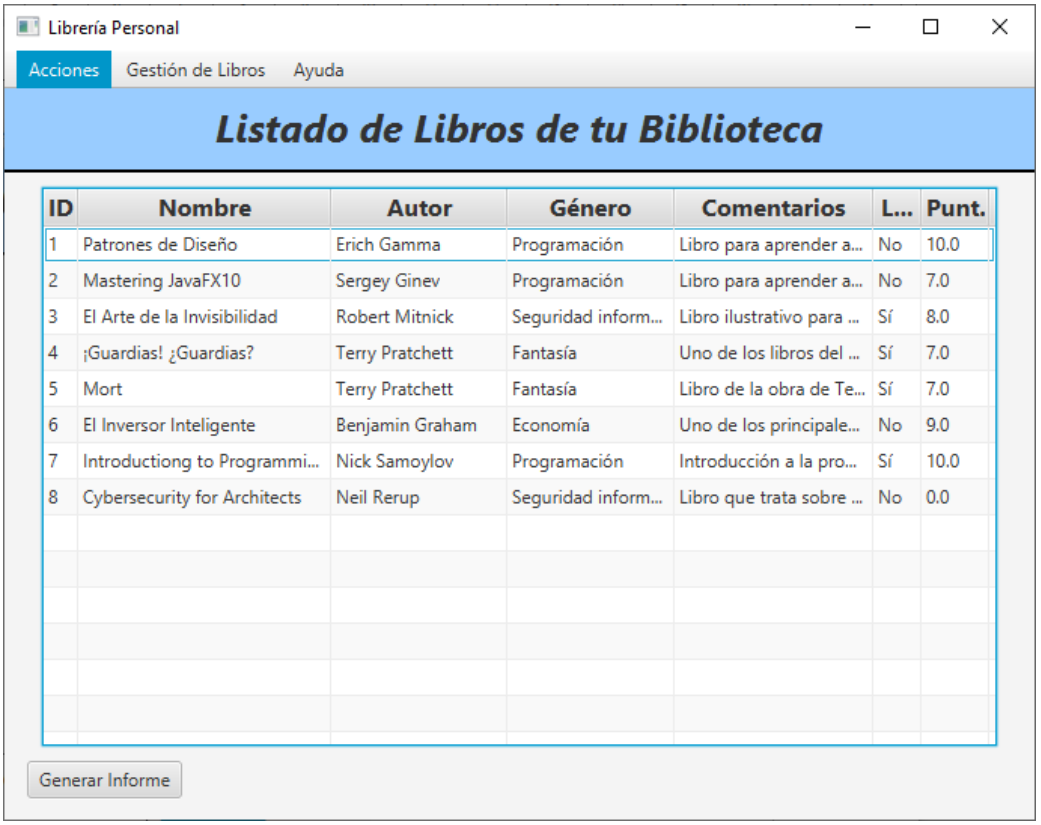
Menú gestión de libros:



Menú Ayuda:



Listar libros:

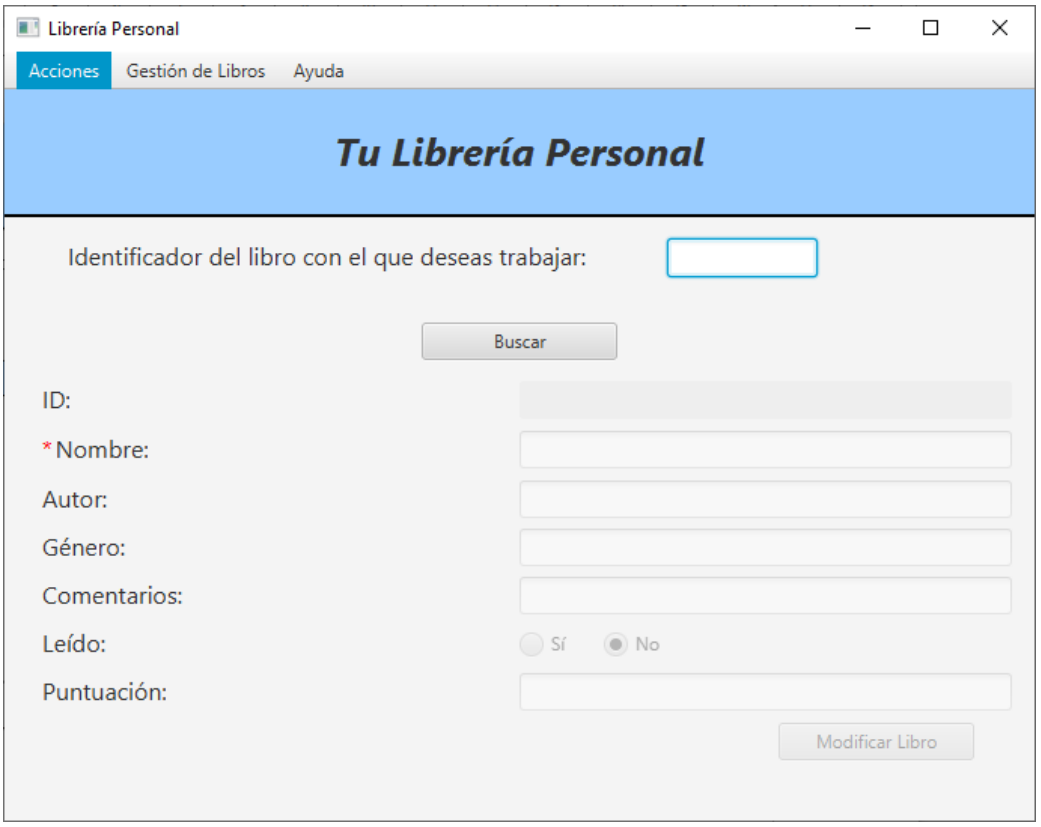


Al darle al botón “Generar Informe”.

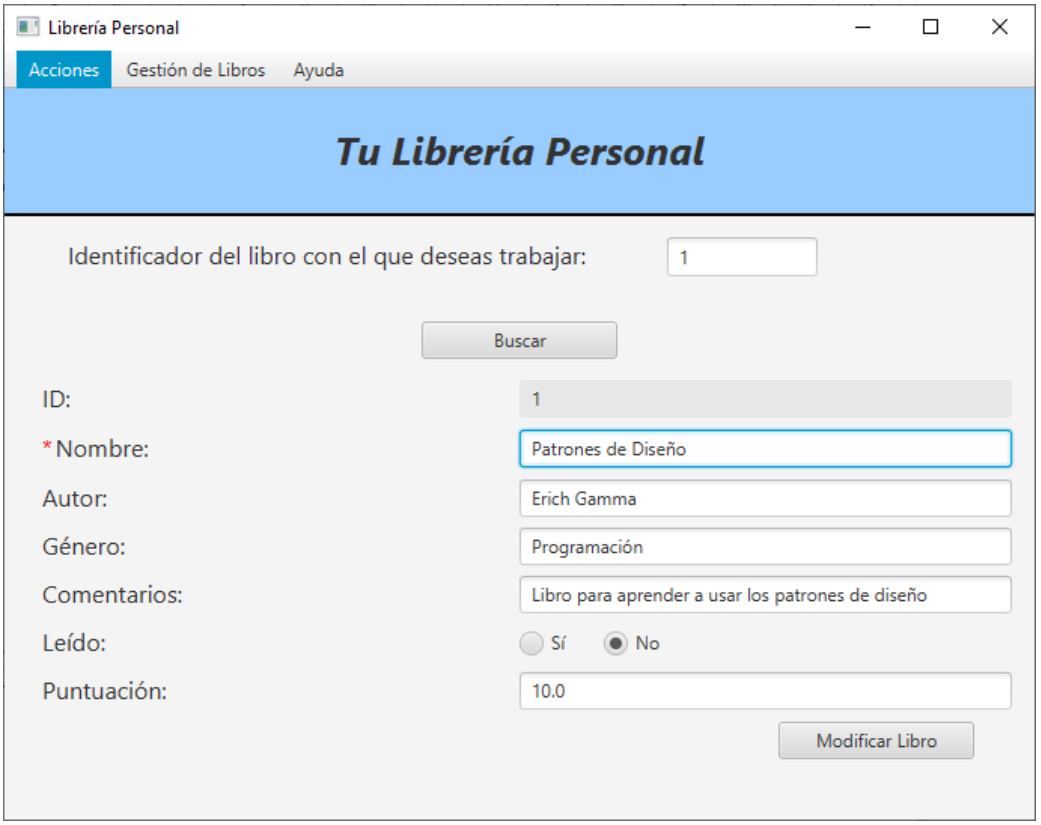
Entrar a “Consultar Libros”

Cuando el usuario indica el id de un libro y le da a buscar:

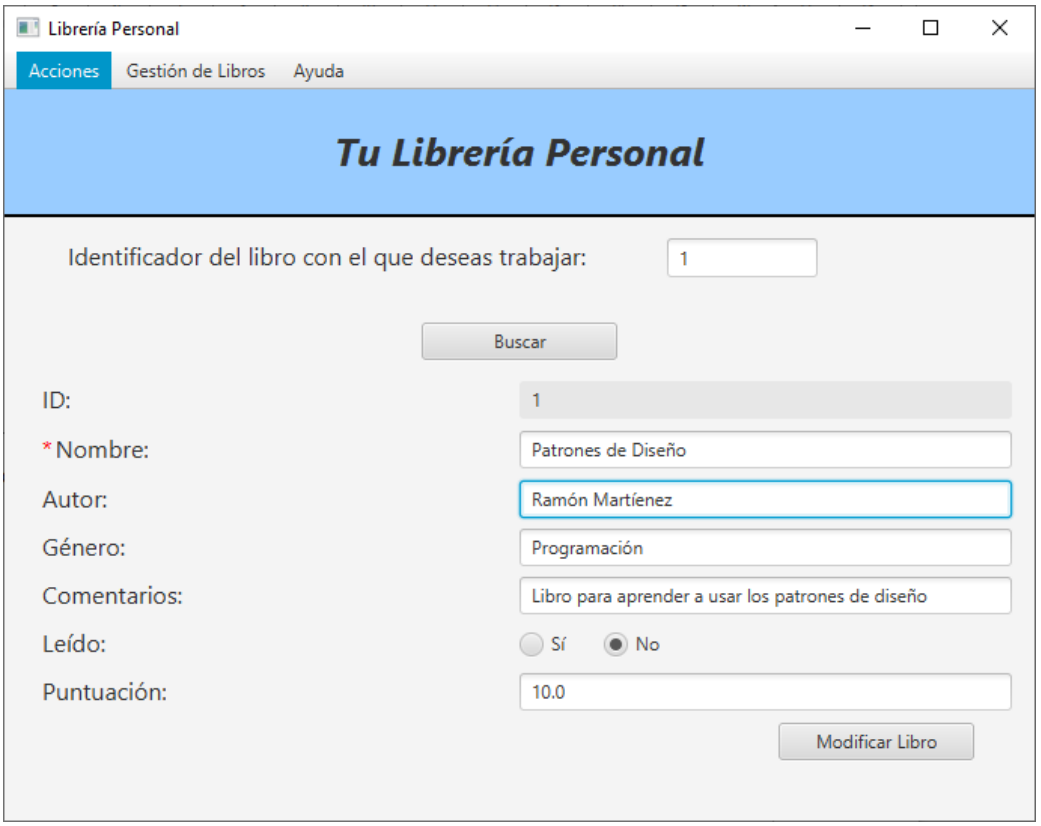
Panel para modificar un libro.



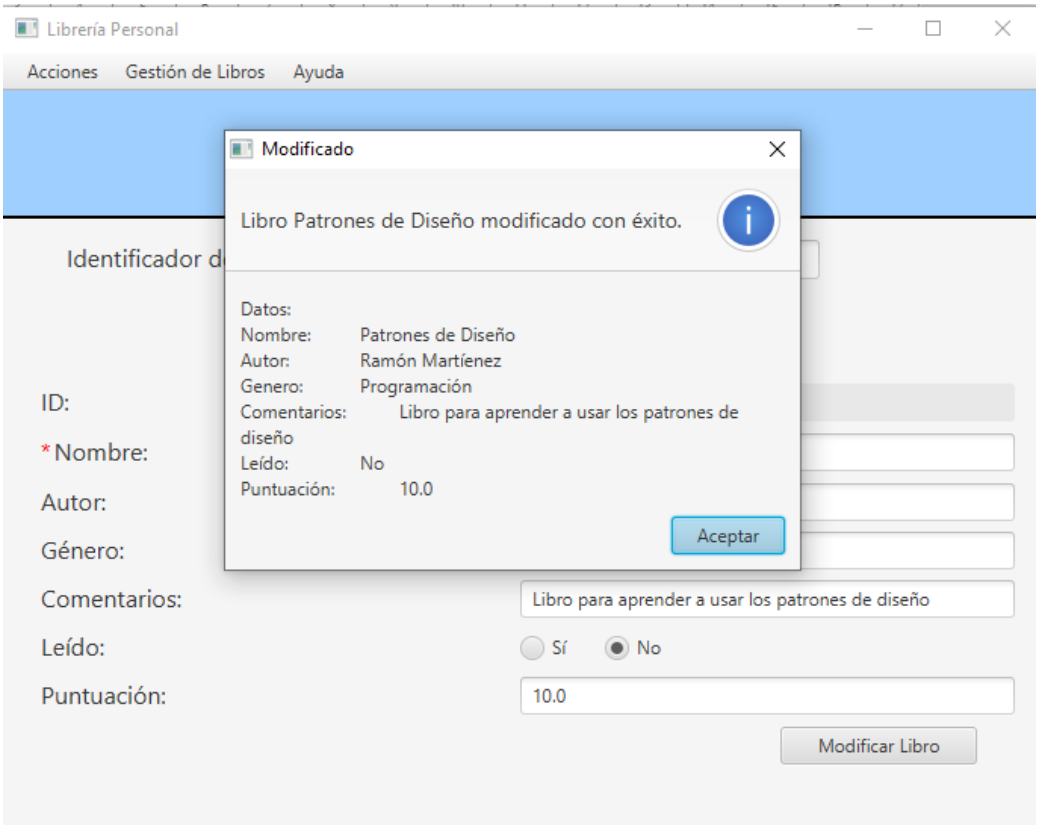
El usuario busca un libro y se habilita la posibilidad de edición.



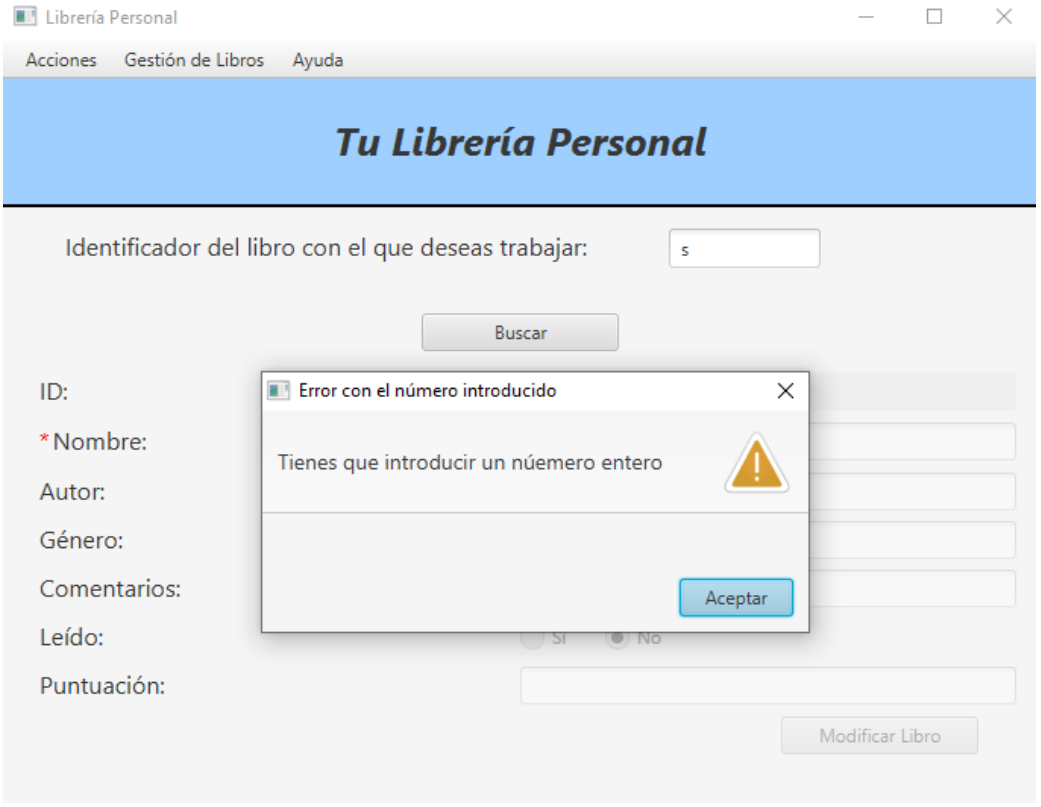
El usuario modifica el libro.



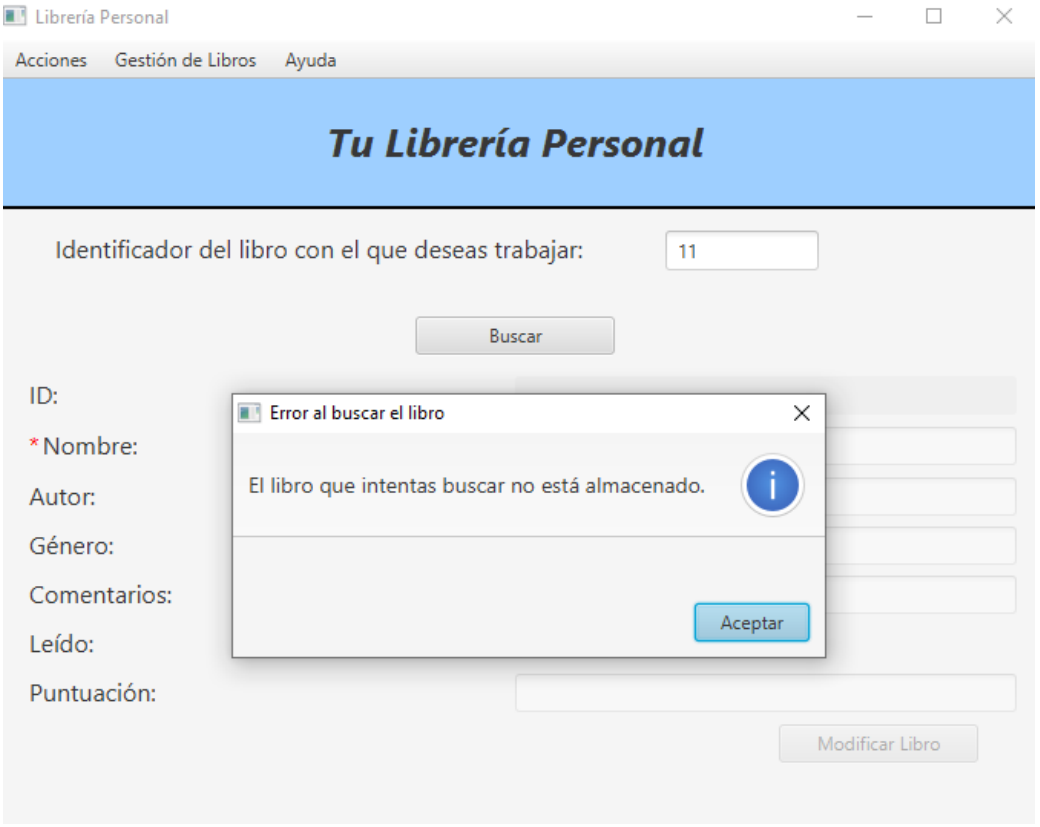
El usuario pulsa “Modificar Libro”



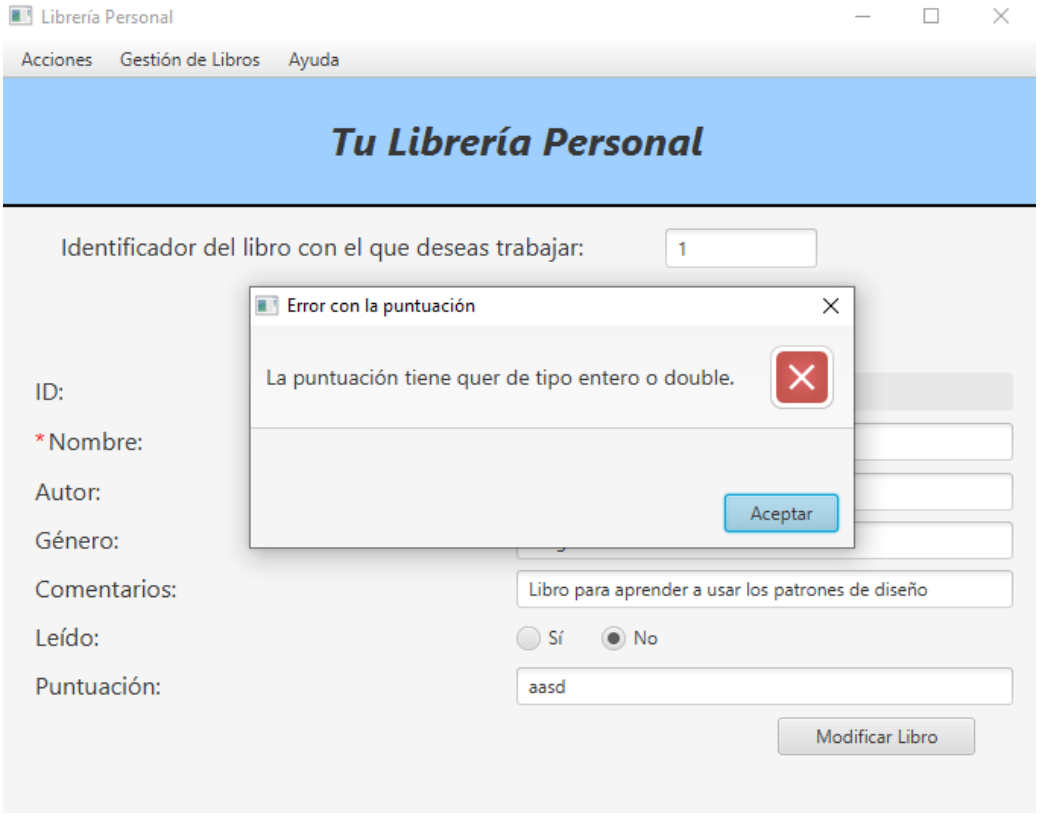
El usuario busca un libro de forma errónea:



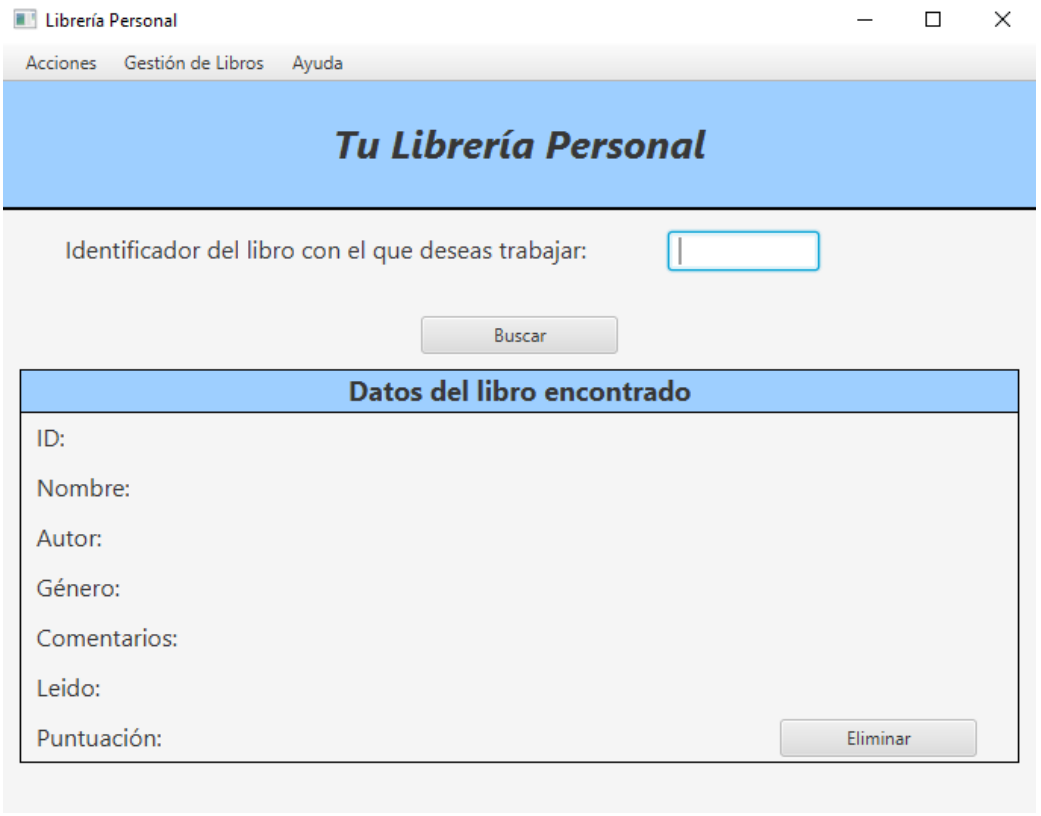
El usuario busca con un identificador válido pero el libro no existe.



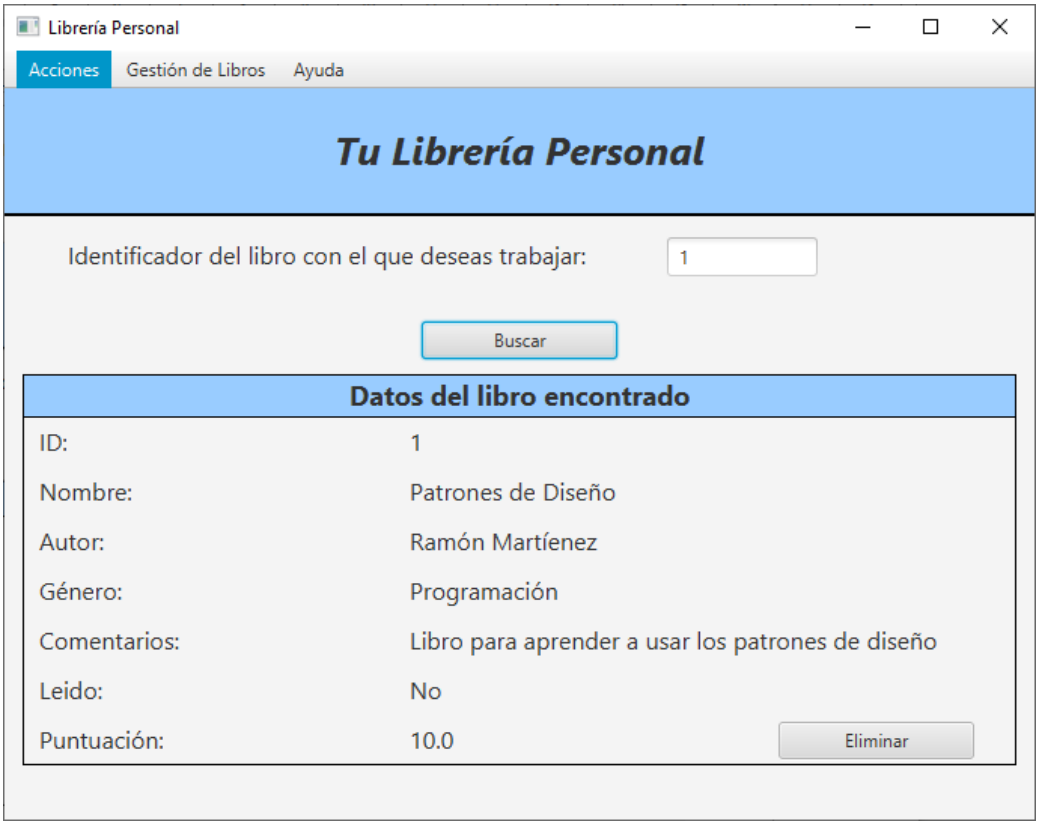
El usuario intenta modificar un libro con una puntuación errónea.



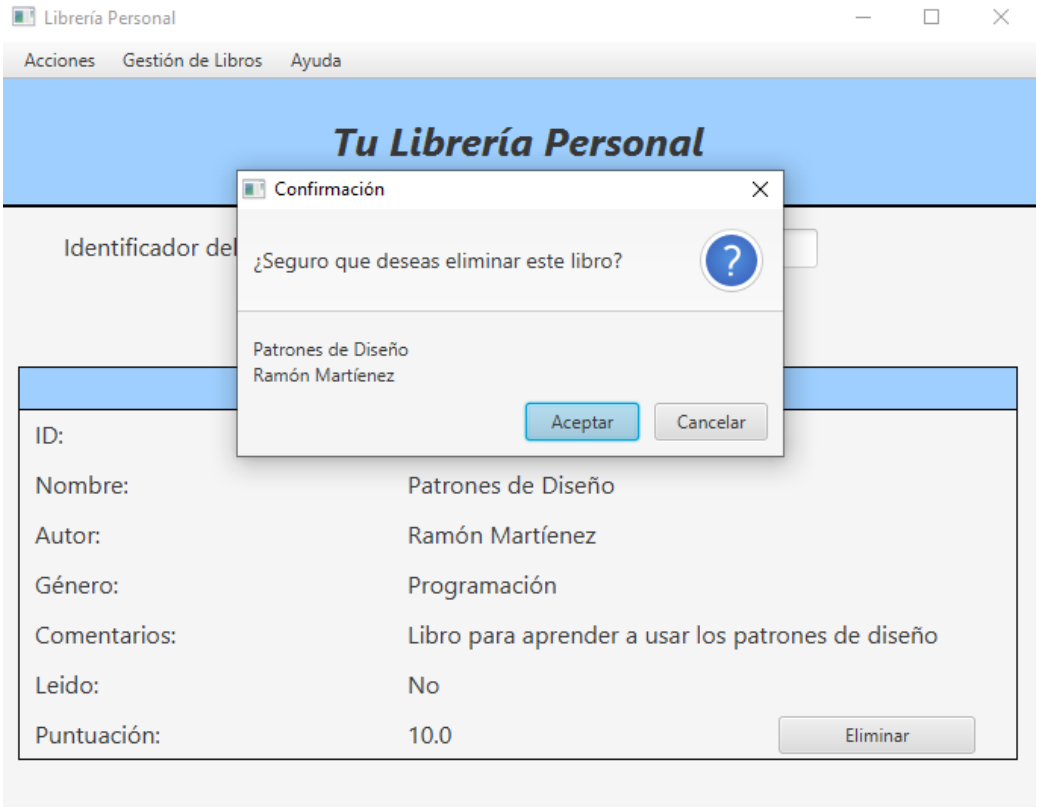
Panel para eliminar un libro.



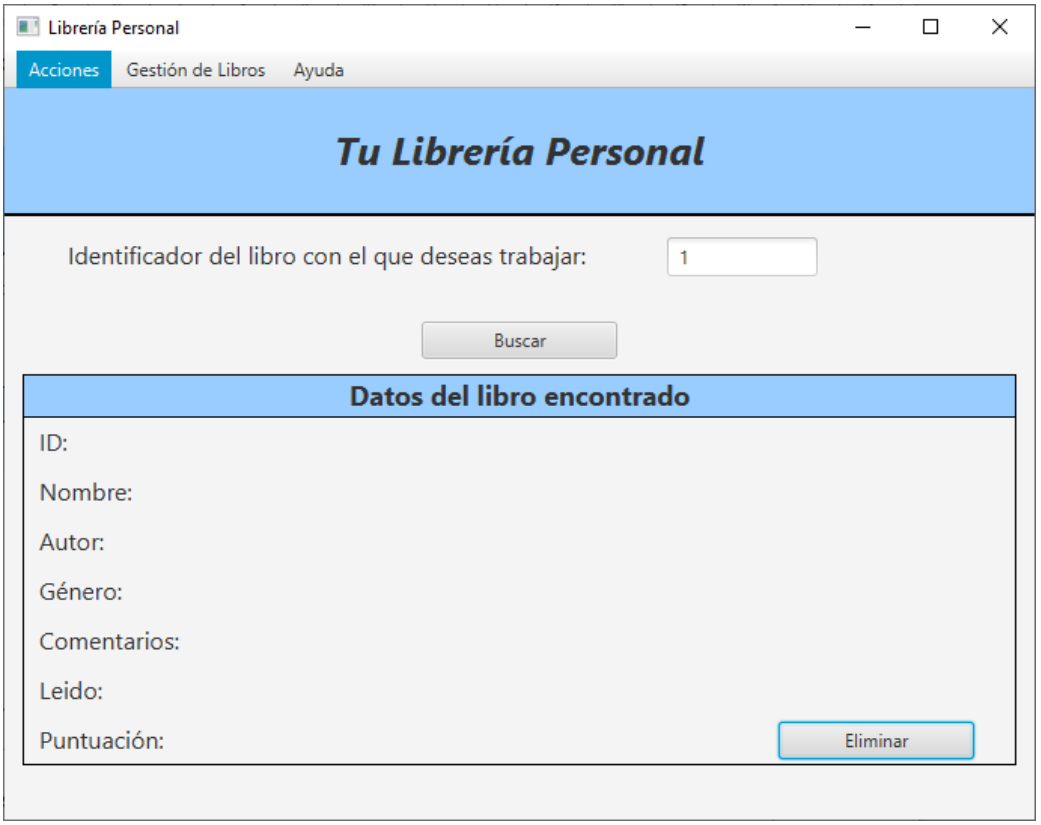
Buscamos un libro a eliminar.



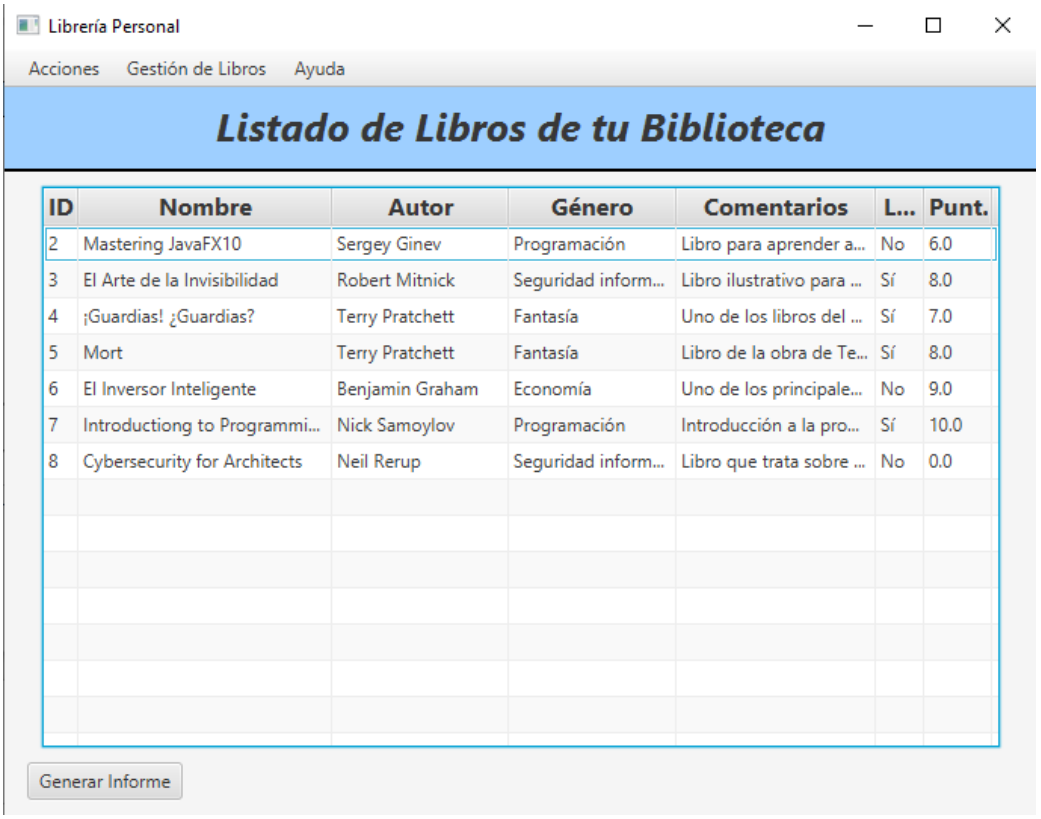
Pulsando el botón eliminar.



Pulsamos Aceptar para confirmar la eliminación.



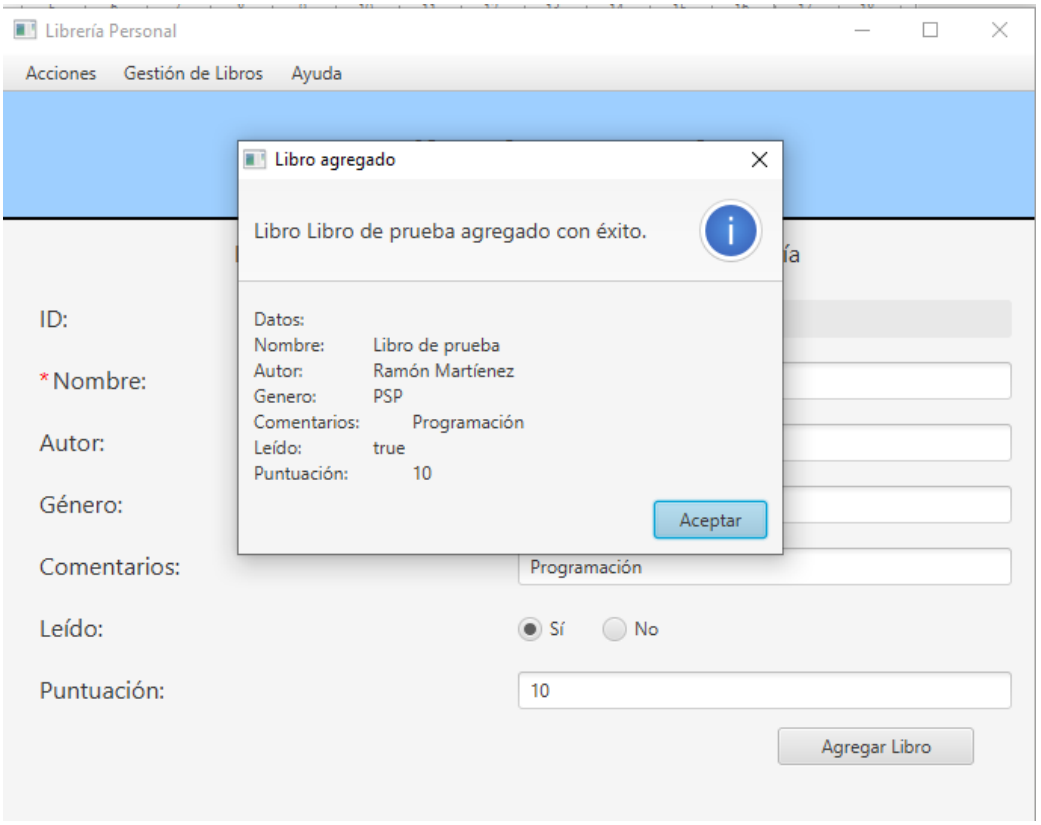
Comprobamos el listado de libros y vemos que el número 1 ya no está y podemos confirmar que se ha eliminado.



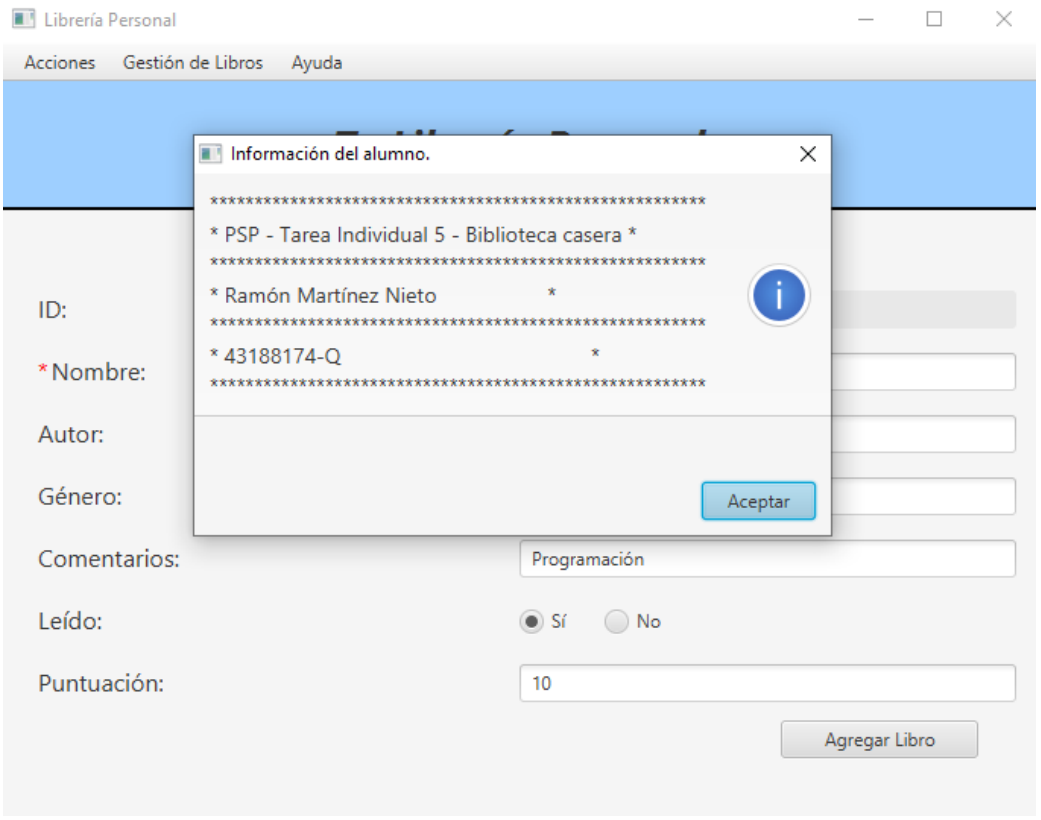
Panel para añadir un libro nuevo.

Relleno los datos.

Al pulsar “Agregar Libro”



Al pulsar Ayuda → Acerca de…



Cuando se pulsa Ayuda → Manual se abre este mismo informe a modo de prueba.