UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2009/2010

Proyecto de Fin de Carrera

CLOC: Un sistema local de gestión interactiva de catálogos

Autor: Rafael López Blanco

Tutores: Diana Pérez Marín

Resumen

El motivo principal de la realización de este proyecto es desarrollar un sistema informático que proporcione la posibilidad de catalogar películas, CDs de música y libros con el ordenador a personas de cualquier edad, siempre y cuando tengan unos conocimientos mínimos de informática. En particular, es necesario que los usuarios sepan utilizar el ratón, y es recomendable que hayan usado alguna vez menús con botones de tipo *Aceptar* o *Cancelar*.

De hecho, esta aplicación está principalmente orientada a personas con poco entrenamiento informático, aunque también podría ser perfectamente utilizada por usuarios informáticos, para que puedan gestionar la organización de películas, CDs de música y libros de una manera sencilla en el ordenador. Esto tiene como ventajas añadidas: evitar el desplazamiento físico y la localización manual de los elementos, con el consiguiente ahorro de tiempo, y aumentar la flexibilidad de distribución de los elementos en el espacio, al mantenerse un índice computerizado de los mismos.

La aplicación se ha implementado en Java, diseñando su interfaz con la clase Swing y gestionando la base de datos con MySQL. Es destacable aquí el esfuerzo realizado para aprender a diseñar interfaces gráficas con Java. Al inicio de este proyecto, se sabía que se quería realizar un sistema usable y por lo tanto, evitar el uso de líneas de comandos que exigen a los usuarios memorizar las instrucciones, o el uso de complejos menús.

Por lo tanto, se decidió realizar un diseño lo más sencillo posible, orientado a las principales funcionalidades de la aplicación y haciendo transparente el resto del proceso a los usuarios. De esta forma, las opciones del menú son:

- Insertar elementos, manteniéndose la consistencia entre las pantallas de inserción de películas, CDs de música y libros en la medida de la posible según aconsejan los principios de usabilidad.

- Modificar los elementos, permitiendo al usuario recuperar los datos que introdujo previamente en el sistema para evitar que los tenga que recordar.

- Borrar elementos para facilitar la renovación de la base de datos desde la interfaz, y sin necesidad de saber MySQL o depender de una persona externa.

- Buscar elementos para cumplir el objetivo de ayudar al usuario en la localización de sus películas, libros y CDs de música.

También se ha incorporado un sistema de ayuda contextual para dirigir a los usuarios durante la realización de las tareas, y se ha proporcionado en el menú la posibilidad de acceder a más información sobre la aplicación en la propia aplicación (sin necesidad de obligar al usuario a leerse un manual externo o buscar ayuda en Internet).

Las pruebas realizadas confirman que la aplicación cumple con su objetivo de poder ser usada por personas con conocimientos mínimos de informática. En particular, en un prueba realizada a 10 personas se ha podido comprobar como gestionar los elementos con el catálogo automatizado les ha parecido sencillo y muy útil.

Índice

Índice de figuras

Índice de Tablas

Agradecimientos

Este proyecto no se habría podido llegar a realizar sin el laborioso esfuerzo de mi tutora Diana Pérez, ya que gracias a ella y a su dedicación de tiempo a este proyecto ayudando tanto en resolución (si estaba en su mano) de dudas, rectificación de errores y sugerencia de posibles alternativas. Pocos tutores como ella hay que dediquen tanto esfuerzo a un proyecto como ella teniendo todo el trabajo que tiene aparte de dirigir los proyectos que tiene a su cargo.
También he de mencionar a Ismael, cotutor ayudante del proyecto, cuya experiencia en el mundo de java me ayudo mucho a la hora del aprendizaje del lenguaje. Por último dar las gracias también a todos los usuarios que se prestaron voluntarios para realizar el apartado de pruebas del proyecto. Sin la ayuda de todos, este proyecto no habría sido posible.
Un saludo a todos y gracias de corazón.

1. Introducción

Actualmente, gran parte de la población acumula un alto número de CDs de música, películas y libros en casa. De hecho, la tendencia es apilarlos en librerías, ocupando espacio y en muchas ocasiones, ocultos tras otros elementos impidiendo un acceso sencillo.

Los expertos en informática pueden crearse una base de datos para gestionar la organización de estos elementos, de forma que tengan un índice que les sirva para registrarlos y localizarlos. Sin embargo, las personas con pocos conocimientos informáticos, desconocen como utilizar un gestor de bases de datos, y sería aconsejable poder proporcionarles una interfaz gráfica sencilla y fácil de manejar.

Por lo tanto, se desarrolló una aplicación informática cuyo objetivo principal es que sea usable, sencilla y útil para tener organizados CDs de música, películas y libros. Así, cualquier usuario, sin necesidad de saber MySQL, simplemente usando el ratón y menús con botones, puede hacer consultas a la aplicación para localizar un determinado título, insertarlo, modificarlo o eliminarlo.

Según Nielsen (2006), una aplicación es **usable** si requiere poco tiempo de aprendizaje, se puede manejar eficiente y eficazmente. Además, debe ser amigable y resultar agradable a los usuarios. Si una característica no se puede utilizar o no se utiliza es como si no existiera.

Con esta idea, se ha desarrollado la aplicación llamada **CLOC**, para que todas las personas puedan gestionar su propio catálogo de forma sencilla y entretenida gracias a la interfaz gráfica. La realización de este proyecto ha sido en Java, con MySQL como gestor de bases de datos.

Se ha escogido Java para utilizar la librería Swing de diseño gráfico. Esto ha supuesto un gran esfuerzo, puesto que en septiembre del año 2009 no se tenían conocimientos ni del paradigma orientado a objetos, ni de Java o cómo realizar interfaces gráficas. No obstante, el objetivo del proyecto requería realizar una interfaz gráfica, puesto que el desarrollo en C por línea de comandos no hubiera sido suficientemente intuitivo, por lo que se dedicó mucho tiempo y esfuerzo al inicio del proyecto, para aprender sobre estas materias y poder desarrollar la aplicación y probarla con usuarios.

La Figura 1 muestra un ejemplo de la interfaz diseñada. En particular, se muestra la posibilidad de inserción de CDs de música en nuestro catálogo. Se quiere destacar como a cada opción se le ha acompañado de un icono gráfico para facilitar a los usuarios la identificación de su funcionalidad, y además se han creado botones grandes para que puedan ser fácilmente utilizados por personas de distintas edades y capacidades físico-mentales.

De hecho, no sólo se ha estudiado la interfaz para intentar que fuera sencilla, sino también los procesos. Así, por ejemplo, una de las tareas que pudiera ser más compleja para personas sin conocimientos informáticos, podría ser buscar la carátula del CD con el botón ‘Examinar’.

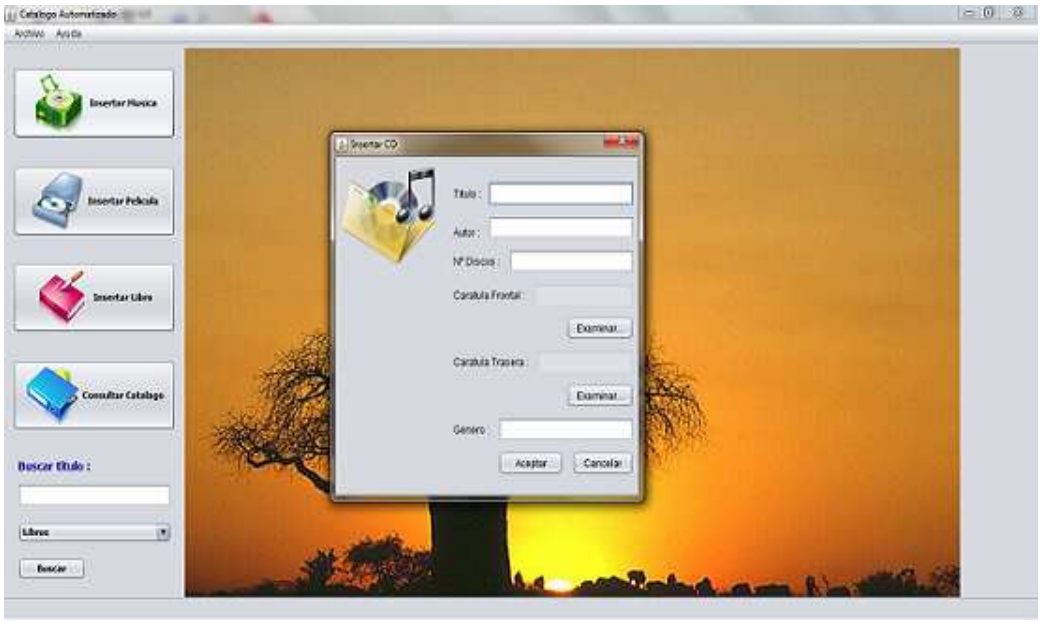

Figura 1. Ejemplo de pantalla de CLOC

Las pruebas realizadas con 10 usuarios han demostrado que se ha conseguido una aplicación vistosa, usable y muy útil para que personas que desean tener bien catalogados CDs de música, libros y películas, puedan hacerlo de una manera flexible y correcta. De hecho, la aplicación está desarrollada para que sea también fácilmente ampliable a otros elementos, como documentales o incluso objetos completamente distintos, como podrían ser zapatos o ropa.

Este documento está dividido en cinco capítulos principales que son los siguientes:

- En el primer capítulo se introduce al lector en la descripción de la aplicación (**CLOC**), una aplicación que pretende ser usable, vistosa y para todos los usuarios, siempre que posean unos conocimientos mínimos de ordenadores. También se habla de la finalidad del proyecto y el porqué de su realización.

- En el segundo capítulo se describen los objetivos principales de la aplicación, y se proporciona la justificación de haber elegido un paradigma de programación orientado a objetos y la metodología empleada.

- En el tercer capítulo se proporciona la descripción informática de la aplicación donde se recogen todos los casos de uso de **CLOC**, la arquitectura de alto nivel con las relaciones entre todas las clases, el diagrama E/R de la base de datos con la explicación de las tablas y los diagramas de todas las clases desarrolladas.

- En el cuarto capítulo se describen las pruebas realizadas como las pruebas unitarias de caja blanca, caja negra, integración, validación con los requisitos, y pruebas de aceptación y de usabilidad con los usuarios para comprobar de forma objetiva si pueden realizar un conjunto de tareas que se les solicita y también saber de forma subjetiva cuál es su opinión.

- En el quinto capítulo se termina el informe con las principales conclusiones extraídas a partir las pruebas realizadas y posible trabajo futuro.

2. Objetivos

2.1 Descripción del problema

El problema a tratar en este proyecto es gestionar un catálogo con los ejemplares de los que disponga un usuario en su casa de manera fácil y sencilla. Para ello, se ha creado **CLOC** una aplicación para catalogar estos elementos de forma usable y amena mediante una interfaz gráfica. En particular, la aplicación puede catalogar tres elementos fundamentales:

- Libros.
- CDs de música.
- Películas.

Además de permitir la inserción de títulos también permite gestionarlos con opciones como:

- Ver títulos ya catalogados de cada elemento.
- Buscar un título utilizando el buscador de la página principal.
- Ver la carátula de un título elegido.
- Eliminar un título del catálogo.
- Modificar campos de títulos ya catalogados, por ejemplo debido a un error al introducir alguna información o por algún cambio por parte de la editorial.

Se pretende sobre todo que los usuarios de **CLOC** no necesiten tener conocimientos técnicos para poder usar la aplicación.

2.2 Requisitos

Los requisitos de CLOC son los siguientes:

1. Usabilidad: CLOC debe poder ser usado con personas sin conocimientos técnicos.

2. Administrador único: CLOC puede contar con un administrador en la base de datos, pero será el propio usuario quien maneje CLOC mediante la interfaz gráfica.

3. Categorías únicas: Cada categoría contará con un identificador único y nombre en la BD. Además, podrá contar con una breve descripción.

4. Identificación de objetos: Cada objeto pertenece a una categoría, y tiene un identificador de objeto, título y localización. Además contará con carátula y una breve descripción.

5. Acceso completo: Los usuarios podrán tener acceso al listado completo de categorías, y objetos existentes clasificados según categorías.

6. Buscador propio: Los usuarios podrán hacer búsquedas a nivel de todas las categorías buscando por nombre del título.

7. Inserción de objetos: Los usuarios podrán dar de alta nuevos objetos en las distintas categorías proporcionando en un formulario la información necesaria para insertarlo en la BD.

8. Modificación de objetos: El usuario podrá modificar cualquier campo de un título introducido en la base de datos ya sea por información errónea o por actualización de la información.

9. Eliminación de objetos: El usuario podrá dar de baja en cualquier momento un objeto creado.

2.3 Estudio de alternativas

A continuación, se realiza el estudio de las distintas alternativas posibles para desarrollar CLOC, y se justifican las elecciones realizadas. Para ello, en primer lugar se revisan los paradigmas de la programación, a continuación los lenguajes orientados a objetos y finalmente, los gestores de bases de datos.

2.3.1 Paradigmas de la programación

Según Louden (2004), se pueden distinguir los siguientes paradigmas de la programación:

2.3.1.1 Programación orientada a objetos

Este tipo de paradigma se basa en la idea de un objeto que vagamente puede describirse como una colección de atributos pertenecientes a un objeto que tiene unos valores en memoria, junto con todas las operaciones que pueden cambiar los valores de dichas localizaciones de memoria.

Un objeto es una variable, con funciones para asignarle un valor y recoger su valor. En muchos lenguajes orientados a objetos, éstos se agrupan en clases que representan todos los objetos con las mismas propiedades. Entonces se crean los objetos como ejemplos particulares, o **instancias**, de una clase. Dentro de la clase definida para un objeto en concreto existen una serie de procedimientos o funciones llamados **métodos** que se encargan de modificar, conseguir o actualizar las características de nuestra instancia del objeto creado.

2.3.1.2 Programación funcional

El paradigma funcional basa la descripción de las computaciones en la evaluación de funciones o en la aplicación de funciones a valores conocidos. Un lenguaje de programación funcional tiene como mecanismo básico la evaluación de una función o **llamada a función**. Esto involucra, además de la propia evaluación de funciones la transferencia de valores como parámetros a funciones y la obtención de valores resultantes como **valores devueltos** de las funciones.

El paradigma funcional no involucra una idea de variable o asignación de variables. En cierto sentido, la programación funcional es lo opuesto a la programación orientada a objetos: se concentra en los valores y funciones en vez de en direcciones de memoria.

También las operaciones repetitivas no se expresan mediante ciclos o bucles (que requieren de variables de control para su terminación), sino mediante funciones recursivas. Podría resultar sorprendente que un lenguaje de programación pueda prescindir completamente de variables y ciclos, pero esto es exactamente lo que hace el paradigma funcional, y al hacerlo ofrece ventajas.

Una de ellas es que le lenguaje se hace mas independiente del modelo de la máquina y que, dado que los programas funcionales se parecen a las matemáticas, resulta más fácil extraer conclusiones precisas con respecto a su comportamiento.

2.3.1.3 Programación imperativa

La programación imperativa es el paradigma de programación más utilizado y mejor desarrollado. Es el paradigma que emergió junto con las primeras computadoras y sus programas en los años cuarenta y sus elementos reflejan directamente las características de arquitectura de las computadoras más modernas.

En la programación imperativa el programa es una serie de pasos, cada uno de los cuáles realizan un cálculo, recupera una entrada o tiene como resultado una salida. La abstracción procedimental es un bloque de creación esencial en la programación imperativa, al igual que las sentencias condicionales, asignaciones, bucles y secuencias. Los lenguajes de programación imperativa principales son Cobol, Fortran, C y C++.

Un lenguaje de programación imperativo es un lenguaje de recorrido completo que además soporta un cierto número de características fundamentales que han nacido con la evolución del paradigma de programación imperativa desde los años cuarenta y cuyas características incluyen (Tucker & Noonan, 2003):

-Tipos de datos para reales, caracteres, booleanos y sus operadores.
-Estructuras de control, bucles for y while, instrucciones case (switch)
-Asignación de elementos y arrays
-Comandos de entrada y salida
-Punteros, procedimientos y funciones.

2.3.1.4 Tabla comparativa

La Tabla 1 muestra la comparativa entre los paradigmas revisados según la dificultad, eficiencia de programación y posibilidades de liberación de memoria. (Louden, 2004) y (Tucker & Noonan, 2003).

	Programación Orientado a Objetos	Programación funcional	Programación Imperativa
Dificultad	Dificultad media	Dificultad sencilla	La mas fácil de aprender
Eficiencia programando	Muy eficiente debido la abstracción y ala programación con clases o módulos	Útil si las funciones declaradas se utilizan muchas veces	Menos eficiente que los otros dos
Liberar memoria	El recolector de basura lo hace solo	Debe liberarla el programador	Debe liberarla el programador

Tabla 1. Comparativa de paradigmas de la programación

2.3.2 Lenguajes orientados a objetos

2.3.2.1 Java

Según Cadenhead & Lemay (2008), Java es un lenguaje orientado a objetos seguro, de plataforma neutra, diseñado para ser más fácil de aprender que C++ y mas difícil de usar incorrectamente que C y C++, aunque no deja de ser un lenguaje que requiere un esfuerzo alto para adaptarse a la POO si se viene de otro tipo de paradigma como la programación estructurada.

Java se preocupa de manera automática de la asignación y desasignacion de la memoria, liberando a los programadores de esta tediosa y compleja tarea. Java no incluye punteros, una potente característica de uso sobre todo para los programadores experimentados que puede ser fácilmente usada de manera incorrecta.

Java incluye una sola herencia en programación orientada a objetos. La falta de punteros y la presencia de gestión automática de la memoria son dos elementos claves en la seguridad de Java.

2.3.2.2 C++

Según Ceballos (2007), C++ es un lenguaje de programación de propósito general basado en el lenguaje de programación C, y ha sido diseñado para ser mucho mejor que C, y soportar la abstracción de datos, la programación orientada a objetos y la programación genérica utilizando plantillas.

El lenguaje C se relaciona normalmente con el sistema operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador de C y la casi totalidad de los programas y herramientas de UNIX fueron escritos en C. Su eficiencia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX.

C++ incorpora una *biblioteca* que fue escrita con la intención de incluir sólo aquellas clases que realmente fueran utilizadas por la mayoría de los programadores. Las facilidades que otorga las podemos resumir en los siguientes puntos:

- Soporte básico, como por ejemplo identificación del tipo de los objetos durante la ejecución y gestión de memoria.
- Soporte proporcionado por la biblioteca de C (manipulación de cadenas, ficheros, etc.)
- La clase **String** para la manipulación de cadenas y las clases para la entrada-salida.
- Clases como contenedor como vectores, listas y mapas.
- Algoritmos de búsqueda y ordenación.

Para la implementación de interfaces, existen distintas posibilidades según el sistema operativo que se utilice. Así, por ejemplo es UNIX existe la clase QT y en Windows se dispone de las MFC.

C++ es, por lo tanto, un lenguaje hibrido que, por una parte, ha adoptado todas las características de la programación orientada a objetos (POO) que no perjudiquen su efectividad, y por otra parte, mejora sustancialmente las capacidades de C. Esto, junto con la biblioteca de clases soportada, dota a C++ de una potencia, eficacia y flexibilidad que lo convierten en un estándar dentro de los lenguajes de programación orientado a objetos.

2.3.2.3 Python

Según Summerfield (2009), Python probablemente sea el lenguaje orientado a objetos de programación más sencillo de aprender y uno de los más fáciles de aprender. El código de Python es claro tanto al leerlo como al escribirlo, y es preciso sin ser complicado. Python es un lenguaje muy expresivo, lo que significa que por regla general podremos escribir bastantes menos líneas de código Python que si estuviésemos escribiendo la misma aplicación en, digamos C++ ó Java.

Python dispone de una librería para diseñar interfaces gráficas, llamada TKinter, con la que se pueden dibujar botones, menús, diálogos, etc..., entre otros elementos.

Python es un lenguaje de plataforma cruzada: en general, el mismo programa Python puede ejecutarse en sistemas Windows y Unix como Linux y Mac OS X, con tan solo copiar el archivo o archivos que compongan el programa en la máquina de destino, sin tener que compilarlo.

Uno de los puntos fuertes de Python es que viene junto a una librería estándar muy completa, esto nos permite hacer cosas como descargar un archivo de Internet, descomprimir un archivo comprimido o crear un servidor Web, todo ello con tan solo unas cuantas líneas de código.

Python se ha creado más recientemente y tiene menos soporte por parte de las comunidades de programadores en la actualidad, además no dispone todavía de la API tan completa de Java.

2.3.2.4 Tabla comparativa

La Tabla 2 muestra la comparativa entre los lenguajes de programación orientada a objetos revisados según dificultad, eficiencia de programación, posibilidades de liberación de memoria y cantidad de información disponible (Cadenhead & Lemay, 2008; Ceballos, 2003; Summerfield, 2009).

	JAVA	**C++**	**PYTHON**
Licencias	Gratuito	Gratuito	Gratuito
Código propietario	Multiplataforma	Depende del entorno donde se programe	Multiplataforma
Curva de aprendizaje	Mas fácil que C++	Difícil de aprender	Sencillo de aprender, pero no tan extendido como Java
Diseño/ Escalabilidad	Permite diseños escalables y orientado a objetos	Permite diseños escalables y orientados a objetos	Permite diseños escalables y orientados a objetos además de programación funcional y estructurada entre otros.

Tabla 2. Comparativa de leguajes orientados a objetos

2.3.3 Gestores de bases de datos

2.3.3.1 Access

Access es una aplicación que forma parte de Microsoft Office y cuyos objetos básicos de la interfaz resultan familiares si se ha utilizado otro producto de Office u otras aplicaciones de Microsoft Windows. Según Training Solutions (2004), Access dispone de más prestaciones que la mayoría de estos programas por lo que familiarizarse con este programa puede resultar algo más complejo.

Los programas de bases de datos sencillas se llaman bases de datos sencillas almacenan la información en una única tabla, que se conoce como archivo plano. Estas bases de datos sencillas se llaman **bases de datos planas**. Sin embargo, Access, puede almacenar los datos en varias tablas relacionadas, llamadas **bases de datos relacionales**, lo que facilita una mejor organización de la información y poder realizar bases de datos más complejas de una manera correcta. Para poderlo utilizar se necesita una licencia de Microsoft.

2.3.3.2 MySQL

Según Pérez (2007), MySQL e**s** un sistema gestor de bases de datos relacionales, que además ofrecer compatibilidad con PHP, Perl, C, HTML, JAVA, etc..., ofrece funciones avanzadas de administración y optimización de bases de datos para facilitar las tareas habituales.

Implementa funcionalidades Web, permitiendo un acceso seguro y sencillo a los datos a través de Internet. Este gestor de base de datos incluye capacidades de análisis integradas, servicios de transformación y duplicación de datos y funciones de programación mejoradas.

Se puede decir que MySQL es un sistema cliente servidor de administración de bases de datos relacionales diseñado para el trabajo tanto en sistemas operativos Windows como en sistemas UNIX/LINUX. Además, determinadas sentencias de MySQL pueden ser embebidas en código PHP y HTML para diseñar aplicaciones Web dinámicas que incorporan la información de las tablas de MySQL a páginas Web. También cabe destacar que MySQL es compatible con software muy potente de diseño Web como Dreamweaver MX.

2.3.3.3 Oracle

Oracle proporciona la capacidad de almacenar y acceder a los datos de forma consecuente con un modelo definido conocido como Modelo relacional (Relational Mode). Por ello Oracle ha sido diseñado como un sistema de gestión de bases de datos relacionales (RDBMS: Relational Database Management System).

Una base de datos en Oracle almacena sus datos en archivos. Internamente existen estructuras de la base de datos que proporcionan una asignación lógica de los datos con los archivos, lo que permite almacenar de forma separada diferentes tipos de datos.

Según Pérez (2002) y Loney (1995), Oracle destaca debido a que tiene una gran capacidad de almacenamiento compaginado con un alto rendimiento sin colapsos cuando la base de datos está bien dimensionada. Además, gestiona muy bien la concurrencia de usuarios, trabaja en Internet y con objetos multimedia y dispone del servidor propio Oracle 9i AS.

La instalación, configuración y administración de Oracle 9i es relativamente sencilla, ya que utiliza aplicaciones llamadas Oracle Universal Installer y Database Configuration Assistant para simplificar estas tareas.

2.3.3.4 Tabla comparativa

La Tabla 3 muestra la comparativa entre los distintos gestores de bases de datos revisados según la dificultad, eficiencia de programación y obtención de licencias (Training Solutions, 2004; Pérez, 2007; Pérez, 2002; Loney, 1995).

	ACCESS	**MySQL**	**ORACLE**
Licencias	Necesita licencias de Microsoft	Es gratuito	Necesita Licencia de Oracle
Código propietario	Solo funciona sobre Microsoft	Multiplataforma	Multiplataforma

Curva de aprendizaje	Fácil de aprender	Dificultad Media	Dificultad Media

Tabla 3 Comparativa de Bases de datos

2.3.4 Toma de decisiones

En este apartado, se indican las razones que justifican haber escogido las siguientes alternativas como se describe a continuación:

1) El paradigma de programación escogido es **POO (Programación Orientada a Objetos)** por las siguientes razones:

-No hace falta liberar memoria.

-Mejor programación gracias a la abstracción, encapsulamiento, polimorfismo y herencia de los datos.

- Paradigma más moderno y con más futuro en la programación.

2) El lenguaje de programación elegido es **Java** debido a ser:

- Código libre.
- Multiplataforma.
- Más fácil de aprender que C++
- Más extendido que Python.

3) El gestor de bases de datos seleccionado es **MySQL** ya que:

- Tiene licencia gratuita.
- Es multiplataforma.
- Trabaja muy bien con información cuyo volumen de datos es medio.

2.4 Metodología empleada

Esta sección describe la metodología empleada para el desarrollo de la aplicación **CLOC**. En particular, se ha seguido el ciclo de vida de software en cascada con iteración, como se muestra en la Figura 2.

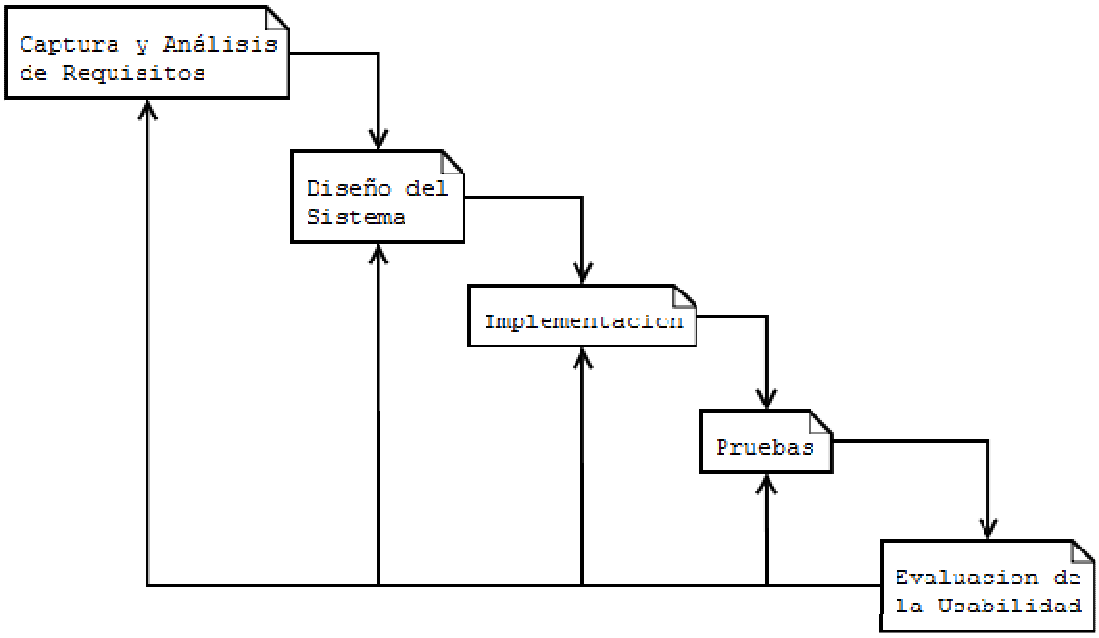

Figura 2. Esquema de ciclo de vida del software

Como podemos ver en la Figura 2 las fases que se han seguido han sido las siguientes:

- Captura y análisis de los requisitos: se analizan los requisitos funcionales y no funcionales de la aplicación.
- Diseño del sistema: se dibuja en papel la aplicación, y su arquitectura de alto y bajo nivel.
- Implementación: se codifica la aplicación integrando todas las clases para ver su interrelación y correcto funcionamiento.
- Pruebas: se prueba que la aplicación cumplan con toda la funcionalidad descritas en los requisitos.
- Evaluación de la usabilidad: se evalúa la sencillez de manejo, utilidad y evaluación del internaza a través de pruebas con usuarios.

Además, como es habitual en el caso del desarrollo orientado a objetos, se utilizará el **Lenguaje de Modelado Unificado** (**UML, Unified Modeling Language**). Según Pressman (2002), **UML** es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que **UML** es un "**lenguaje de modelado**" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir.

3. Descripción informática

3.1 Especificación

En esta sección, se describen los requisitos de CLOC tanto funcionales como no funcionales, y se proporcionan los casos de uso realizados durante el análisis de la aplicación para modelar los requisitos de **CLOC** desde el punto de vista del usuario.

3.1.1 Análisis de los Requisitos

Los requisitos funcionales especifican los servicios que debe proporcionar la aplicación para que resulte aceptable. En el caso de CLOC son los siguientes:

- **REQ1**: El usuario tendrá la posibilidad de insertar CDs de música, películas y libros.
- **REQ2:** El usuario tendrá la posibilidad de consultar tanto películas, como CDs de música como libros tenga almacenados en CLOC.
- **REQ3:** El usuario tendrá la posibilidad de buscar un libro, película o CD de música utilizando el buscador que utiliza la aplicación en lugar de consultar el catálogo directamente por lo que resultará más sencillo encontrar un título.
- **REQ4:** El usuario tendrá la posibilidad de poder eliminar un libro, película o CD de música del catalogo en cualquier momento.
- **REQ5:** El usuario podrá modificar cualquier campo de un libro, película o CD de música ya sea por modificación de una equivocación al insertar el título o por actualización del mismo titulo.
- **REQ6:** El usuario tendrá la posibilidad de ver las carátulas de los CDs de música, películas y libros en cualquier momento gracias al visor de carátulas.

Los requisitos no funcionales especifican los niveles de usabilidad, rendimiento, tiempo, confiabilidad, y características de la interfaz necesarios para que la aplicación sea aceptable. En el caso de CLOC son los siguientes:

- **REQ1NF**: **CLOC** contará con un sistema de interfaz ameno y usable para favorecer la comunicación con el usuario.
- **REQ2NF**: **CLOC** gestionará las acciones del usuario proporcionando un rendimiento óptimo de una forma ágil cumpliendo con las restricciones de tiempo.
- **REQ3NF**: **CLOC** podrá utilizarse desde cualquier ordenador y cualquier sistema operativo siempre que el ordenador cuente con su propia máquina virtual de JAVA para ejecutar la aplicación.

3.1.2 Casos de uso

Se define **actor** como cualquier agente externo que llegue a producir una interacción con la aplicación (**CLOC**). Por tanto, un **actor** será toda entidad externa al sistema que guarda una relación con este y que le demanda una funcionalidad. En esta aplicación, se distingue un actor que es el usuario, y que se puede definir como la persona que va a interactuar con

CLOC, realizando operaciones como insertar, consultar, buscar, modificar, eliminar o ver la carátula de un CD de música, película o libro.
A continuación, se muestran varios ejemplos de casos de uso de CLOC.

Caso de uso "Entrada a la aplicación"

Actores: Usuario.
Descripción: Se permite el acceso al usuario.

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. El caso de uso comienza cuando el usuario hace doble clic sobre el icono de la aplicación.	2. Se inicia la aplicación y la aplicación muestra su interfaz con el menú de acciones.

Tabla 4. Caso de uso "Entrada a la aplicación"

Caminos alternativos:
Evento 2: El usuario cierra la aplicación.

Caso de uso "Insertar un libro"

Actores: Usuario.
Descripción: El usuario inserta un libro en el catalogo **CLOC.** De hecho, en general, este caso de uso es el mismo independientemente del objeto que se inserte para dar consistencia a la aplicación. Se muestra el ejemplo con un libro, pero sería igual para el caso de un CD de música o una película.

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. El caso de uso comienza cuando el usuario con la aplicación una vez iniciada, decide presionar el botón "Insertar un Libro".	2. El sistema responde generando un formulario para un libro, a rellenar por el usuario.
3. El usuario rellena el formulario y hace clic en el botón Aceptar:	4. El sistema guarda el libro en el catálogo correspondiente.
	5. El sistema cierra el formulario.

Tabla 5 Caso de uso "Insertar un Libro"

Caminos alternativos:
Evento 3: El usuario cierra el formulario.
Evento 4: No se han rellenado correctamente algunos campos y se vuelve a 3

Caso de uso "Consultar Catálogo de Películas"

Actores: Usuario.
Descripción: El usuario consulta todo el catalogo de películas de **CLOC.** De hecho, este caso de uso es el mismo independientemente del catálogo que se consulte.

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. El caso de uso comienza cuando el usuario con la aplicación una vez iniciada, decide presionar el botón "Consultar Catalogo".	2. El sistema lanza un menú para que el usuario elija el catálogo que desea ver.
3. El usuario selecciona el botón de "Ver Películas".	4. El sistema obtiene todas las películas del catalogo.
	5. El sistema muestra todas las películas con todas sus características en la interfaz.
6. El usuario consulta el catálogo y observa las películas que dispone.	

Tabla 6 Caso de uso "Consultar Catálogo de libros"

Caminos alternativos:
Evento 3: El usuario cierra el menú ya que no decide consultar ningún catálogo.

Caso de uso "Buscar un CD de Música"

Actores: Usuario.
Descripción: El usuario busca un CD de música utilizando el buscador de la aplicación. Igual que en los casos de uso anteriores, el caso de uso de "Buscar un CD de Música" es el mismo que para el de "Buscar una Película" y "Buscar un Libro".

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. El caso de uso comienza cuando el usuario con la aplicación una vez iniciada, se sitúa en esquina inferior izquierda e introduce el nombre del CD de música.	
2. El usuario selecciona del menú desplegable que quiere buscarlo en el catalogo de CDs de música y presiona Aceptar.	

	3. El sistema obtiene todos los CDs de música que tenga con ese nombre o parecido.
	4. El sistema muestra todos los CDSs que tenga con ese nombre o parecido en la interfaz gráfica.

Tabla 7. Caso de uso "Buscar un CD de Música"

Caminos alternativos**:**
Evento 2: El usuario decide no buscar el CD.
Evento 3: El sistema muestra un mensaje al usuario diciéndole que el CD que busca no se encuentra en el catalogo.

Caso de uso "Modificar un CD de Música"

Actores: Usuario.
Descripción: El usuario desea modificar un campo (título, carátula frontal, carátula trasera…) de un CD de música utilizando el visor de catálogos o el buscador según le resulte más fácil al usuario. El caso de uso de "Modificar un CD de Música" es el mismo que para el de "Modificar una Película" y "Modificar un Libro".

ACCION DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. El caso de uso comienza donde termina el caso de uso "Buscar un CD de música" o donde termina el caso de uso "Ver catalogo de CDs".	
2. El usuario selecciona *Id _ objeto* del catalogo de CDs de música correspondiente al CD del cual quiere modificar un campo.	
3. El usuario presiona el botón de "Modificar"	4. El sistema responde mostrando por pantalla una ventana para modificar un campo del CD seleccionado.
5. El usuario rellena el campo correspondiente y selecciona del menú desplegable que campo quiere modificar	
6. El usuario pulsa aceptar para modificar el CD.	7. El sistema responde mostrándole un mensaje por pantalla diciéndole que el CD se ha modificado con éxito.

Tabla 8. Caso de uso "Modificar un CD de música"

Caminos alternativos**:**
Evento 2: El usuario no ha seleccionado un *Id _ objeto* y al pulsar el botón de "Modificar" el sistema le muestra un mensaje por pantalla que debe seleccionar el *Id _ objeto* del titulo que quiera modificar.
Evento 5: El usuario se ha confundido de CD y decide cerrar la ventana ya que de este no quiere modificar nada.

Caso de uso "Eliminar un CD de Música"

Actores: Usuario.
Descripción: El usuario desea eliminar un CD de música utilizando el visor de catálogos o el buscador según le resulte más fácil al usuario. El caso de uso de "Eliminar un CD de Música" es el mismo que para el de "Eliminar una Película" y "Eliminar un Libro".

ACCION DEL ACTOR	**RESPUESTA DEL SISTEMA**
1. El caso de uso comienza donde termina el caso de uso "Buscar un CD de música" o donde termina el caso de uso "Ver catalogo de CDs".	
2. El usuario selecciona *Id _ objeto* del catalogo de CDs de música correspondiente al CD del cual quiere eliminar.	
3. El usuario presiona el botón de "Eliminar"	4. El sistema responde mostrando por pantalla el mismo catálogo pero sin todos los datos del CD eliminado.

Tabla 9. Caso de uso "Eliminar un CD de música"

Caminos alternativos**:**

Evento 2: El usuario no ha seleccionado un *Id _ objeto* y al pulsar el botón de "Eliminar" el sistema le muestra un mensaje por pantalla que debe seleccionar el *Id _ objeto* del titulo que quiera eliminar.

Caso de uso "Ver carátula frontal de un CD de Música"

Actores: Usuario.
Descripción: El usuario desea ver la carátula frontal de un CD de Música utilizando el visor de catálogos o el buscador según le resulte más fácil al usuario. El caso de uso de "Ver Carátula Frontal de un CD de música" es el mismo que para el de "Ver carátula de una Película" y "Ver carátula de un libro" ya que películas y libros solo cuenta con la portada principal.

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. El caso de uso comienza donde termina el caso de uso “Buscar un CD de música” o donde termina el caso de uso “Ver catalogo de CDs”.	
2. El usuario selecciona *Id _ objeto* del catalogo de CDs de música correspondiente al CD del cual quiere ver la carátula frontal.	
3. El usuario presiona el botón de “Ver Carátula Frontal”	4. El sistema responde mostrando por pantalla una ventana donde se puede apreciar la carátula frontal del CD seleccionado.

Tabla 10. Caso de uso “Ver Carátula Frontal un CD de música”

Caminos alternativos**:**

Evento 2: El usuario no ha seleccionado un *Id _ objeto* y al pulsar el botón de “Ver Carátula Frontal” el sistema le muestra un mensaje por pantalla que debe seleccionar el *Id _ objeto* del CD del que quiera ver la carátula.

3.1.3 Diagramas de casos de uso

Insertar un objeto en el catálogo

La Figura 3 muestra el diagrama con las acciones que debe hacer el usuario para insertar un objeto en el catálogo **CLOC**.

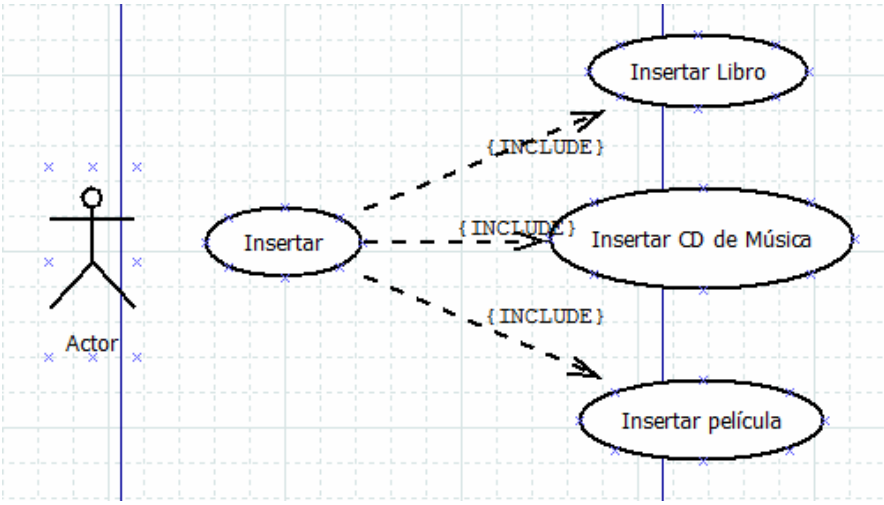

Figura 3. Caso de uso “Insertar un objeto en el catálogo”

Consultar catálogo

La Figura 4 muestra las distintas acciones que tiene el usuario cuando quiere consultar un objeto en el catálogo **CLOC**.

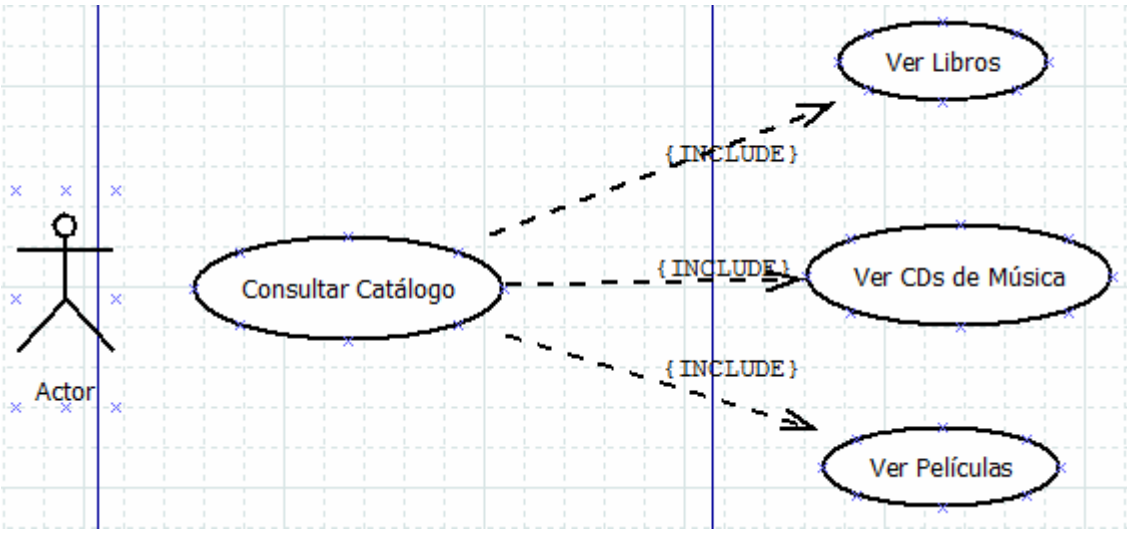

Figura 4. Caso de uso "Consultar catálogo"

Buscar un libro, CD de música o película

La Figura 5 muestra las distintas acciones que tiene el usuario cuando quiere buscar un libro, película o CD de música en el catálogo **CLOC**.

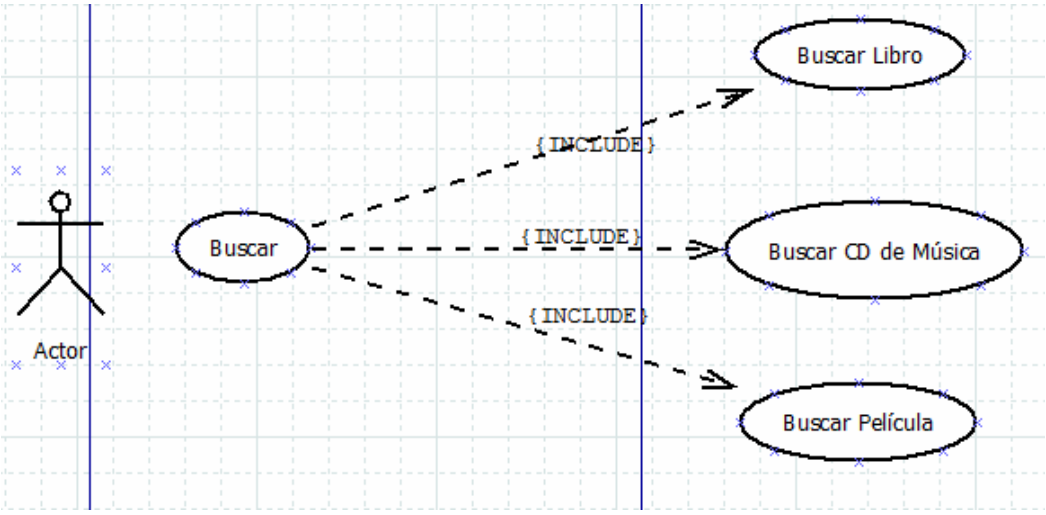

Figura 5. Caso de uso "Buscar un libro, película o CD de música"

Modificar un libro, CD de música o película

La Figura 6 muestra las distintas acciones que tiene el usuario cuando quiere modificar un libro, película o CD de música en el catálogo **CLOC**.

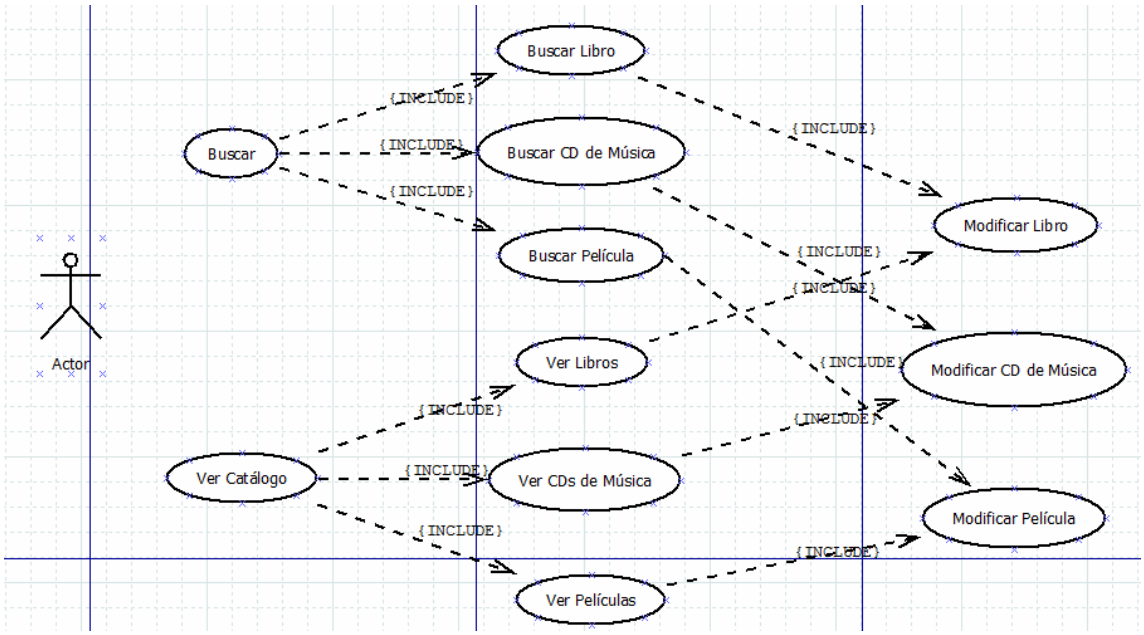

Figura 6. Caso de uso “Modificar un libro, película o CD de música”

Eliminar un libro, CD de música o película

La Figura 7 muestra las distintas acciones que tiene el usuario cuando quiere eliminar un libro, película o CD de música en el catálogo **CLOC**.

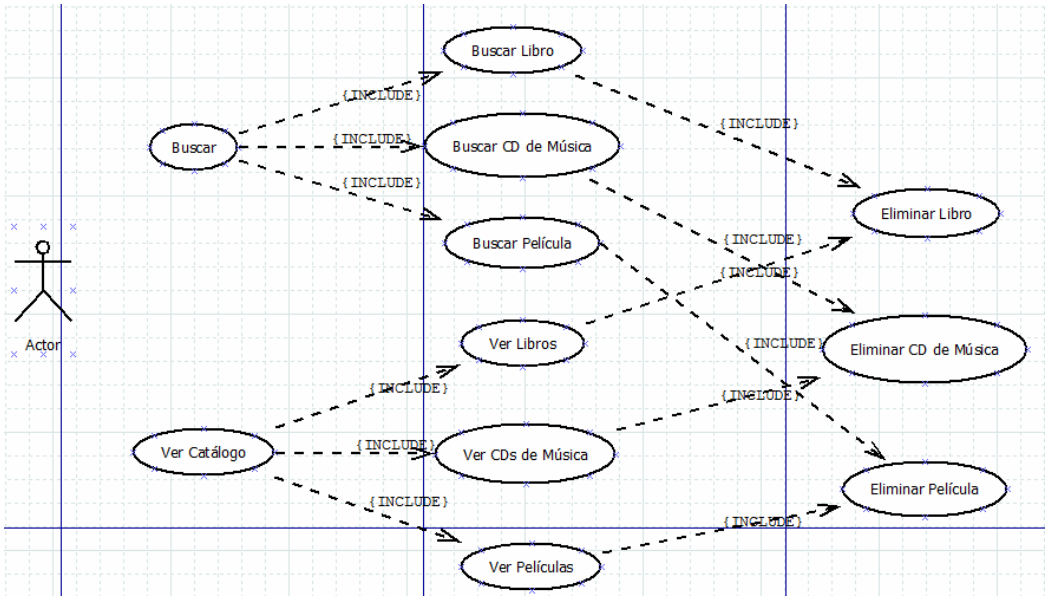

Figura 7. Caso de uso Eliminar un libro, película o CD de música.

Ver Carátula un libro, CD de música o película

La Figura 8 muestra las distintas acciones que tiene el usuario cuando quiere ver la carátula un Libro, Película o CD de Música en el catálogo **CLOC**.

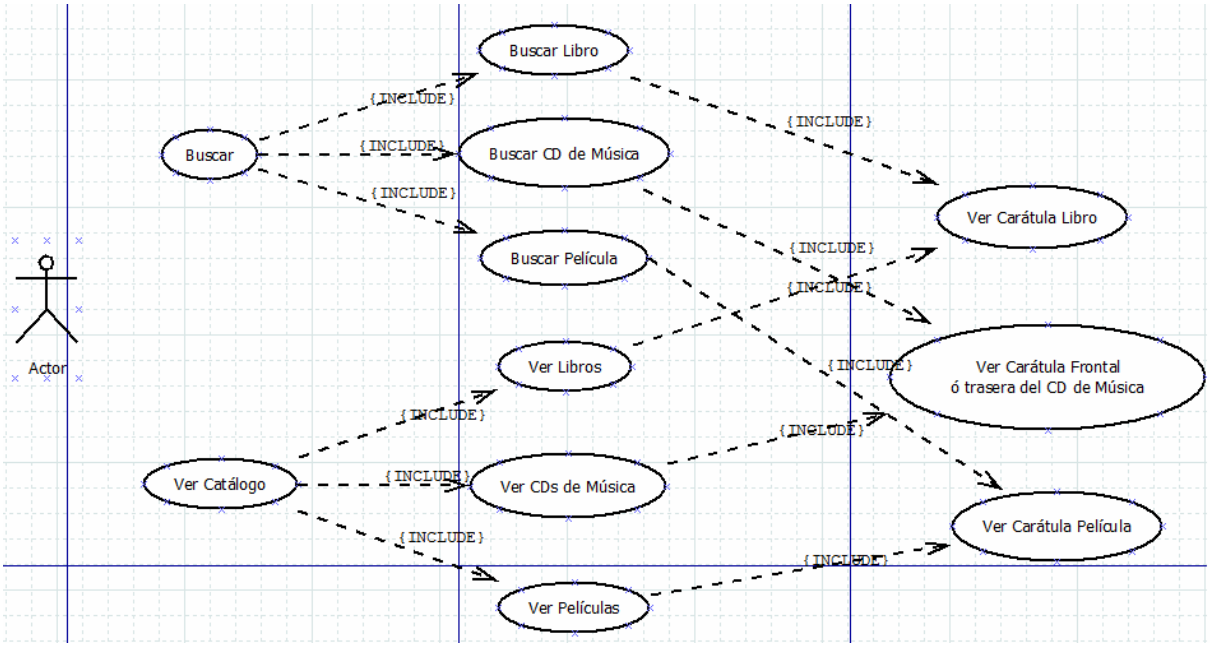

Figura 8. Caso de uso Ver Carátula de un libro, película o CD de música.

3.1.4 Diagramas de interacción

Descripción de las clases de análisis

Clase límite o interfaz: se encarga de modelar la interacción entre los actores y el sistema. Representa la interfaz del sistema pero solo a nivel conceptual. Describe la información presentada al actor y las peticiones que hace el actor al sistema.

Clase control: representa la relación lógica y coordinación entre objetos y encapsulan el flujo de control de un determinado caso de uso. Solo se encarga de la lógica del sistema, por lo que ni interacciona con el actor ni almacena información.

Clase entidad: se utiliza para mantener la información de manera permanente la cual se encuentra relacionada con un caso de uso, muestran una estructura de datos lógica y ayudan a entender que información manipular.

En los siguientes diagramas se mostrara la relación existente entres estas clases y los usuarios de la aplicación.

Diagrama de interacción para el caso de uso "Insertar Libro"

La Figura 9 muestra la interacción para una petición de insertar un libro.

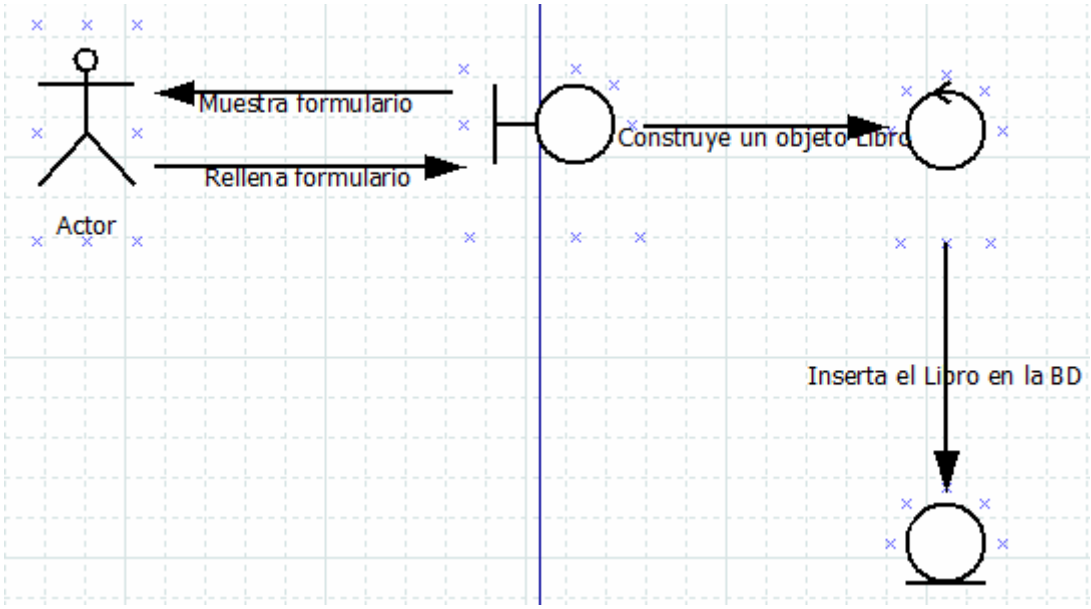

Figura 9. Diagrama de interacción de caso de uso "Insertar Libro"

Diagrama de interacción para el caso de uso "Ver Catálogo de Películas"

La Figura 10 muestra la interacción para ver el catálogo de películas.

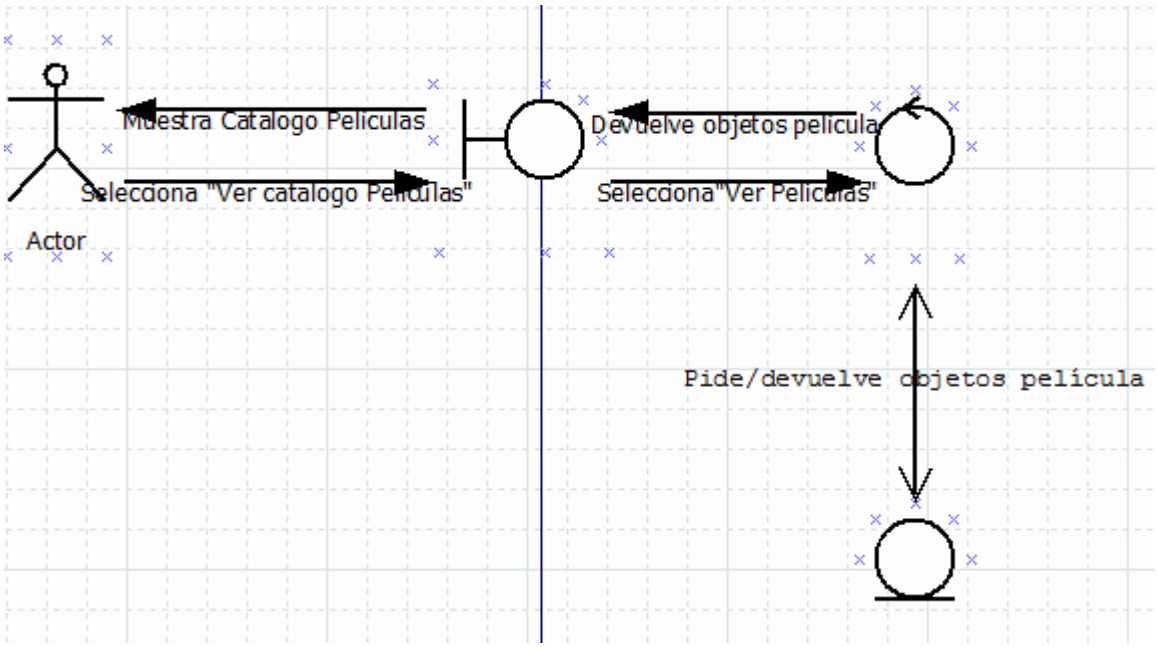

Figura10. Diagrama de interacción de caso de uso “Ver catálogo de películas”

Diagrama de interacción para el caso de uso “Buscar CD de Música”

La Figura 11 muestra la interacción para buscar un CD de música.

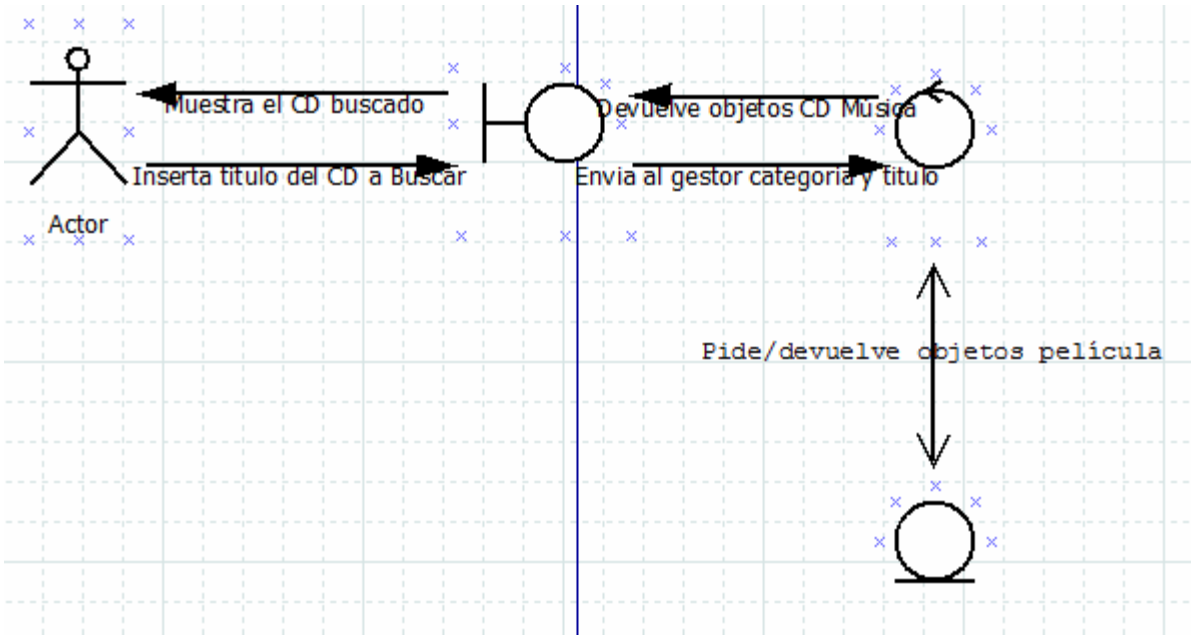

Figura 11. Diagrama de interacción de caso de uso “Buscar un CD de Música”

Diagrama de interacción para el caso de uso “Modificar CD de Música”

La Figura 12 muestra la interacción para modificar un campo de un CD de música.

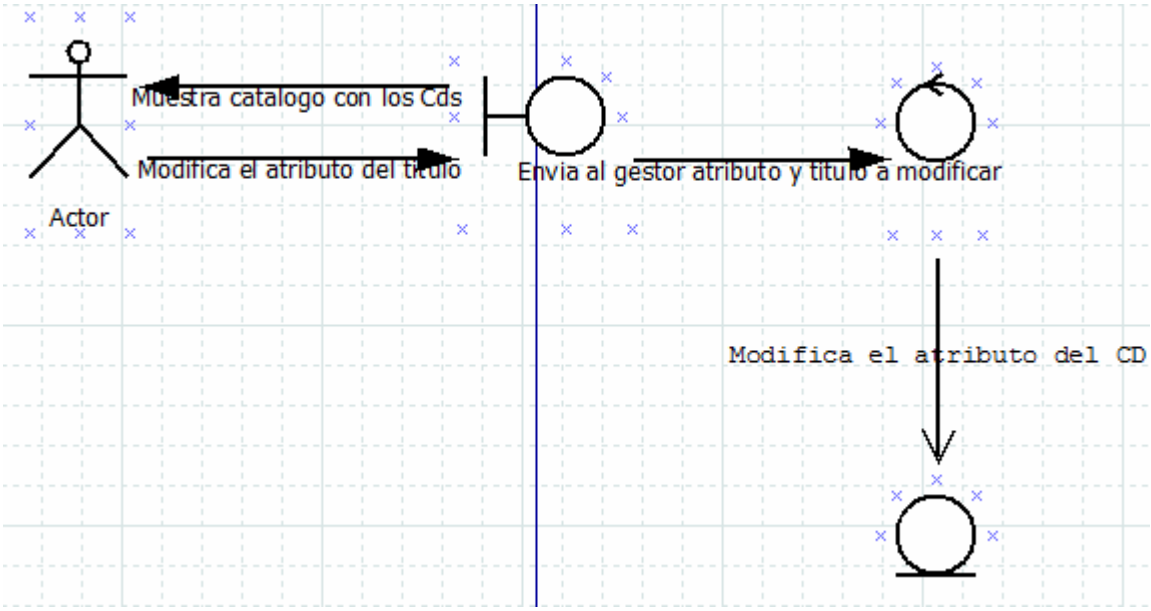

Figura 12. Diagrama de interacción de caso de uso “Modificar un CD de Música”

Diagrama de interacción para el caso de uso “Eliminar CD de Música”

La Figura 13 muestra la interacción para una petición de eliminar un CD de música.

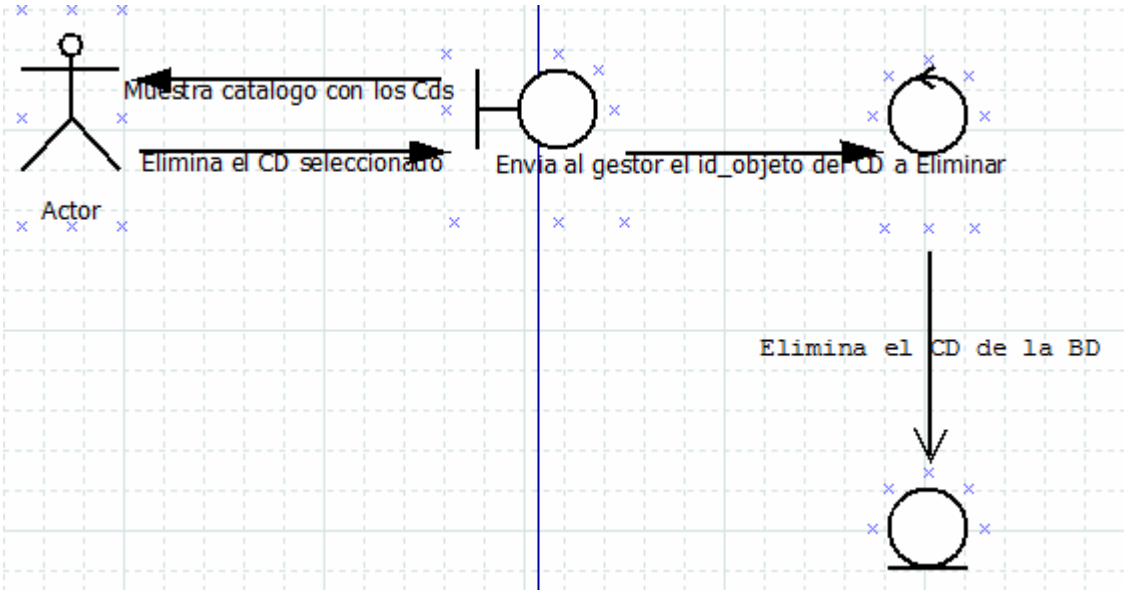

Figura 13. Diagrama de interacción de caso de uso “Eliminar un CD de Música”

Diagrama de interacción para el caso de uso “Ver Carátula Frontal de un CD de Música”

La Figura 14 muestra una interacción usuario-máquina para una petición de ver la carátula frontal de un CD de música.

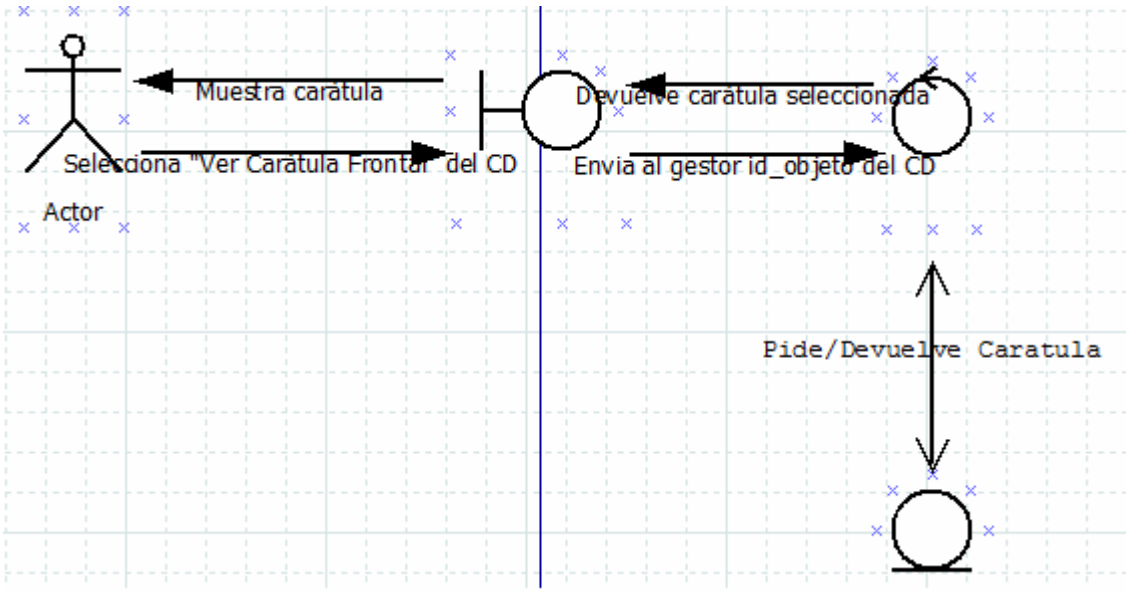

Figura 14. Diagrama de interacción de caso de uso “Ver Carátula Frontal de un CD de Música”

3.2 Arquitectura de alto nivel

La Figura 15 muestra la arquitectura de alto nivel reflejada por un diagrama de flujo donde cada caja representa una clase de la aplicación y las flechas entre ellas la acción que las relaciona.

Como se puede apreciar en la Figura 15, la clase más importante es la **ControladorBD.java** (en color negro), ya que es la clase con la que se relacionan la mayoría del resto de las clases. En color azul se representa la acción de eliminar un objeto de la BD. En color rojo se representa la inserción de objetos en la BD y mostrar las carátulas (este color rojo solo es utilizado en **CatalogoPelicula1.java, CatalogoLibro2.java y CatalogoMusica1.java**; para **CatalogoPelicula.java, CatalogoLibro1.java y CatalogoMusica.java** la acción de ver la carátula es en azul). En color verde se representa la acción de Modificar un campo de un objeto (excepto para **CatalogoPelicula.java, CatalogoLibro1.java y CatalogoMusica.java** que es en negro). El resto de las acciones es en color negro.

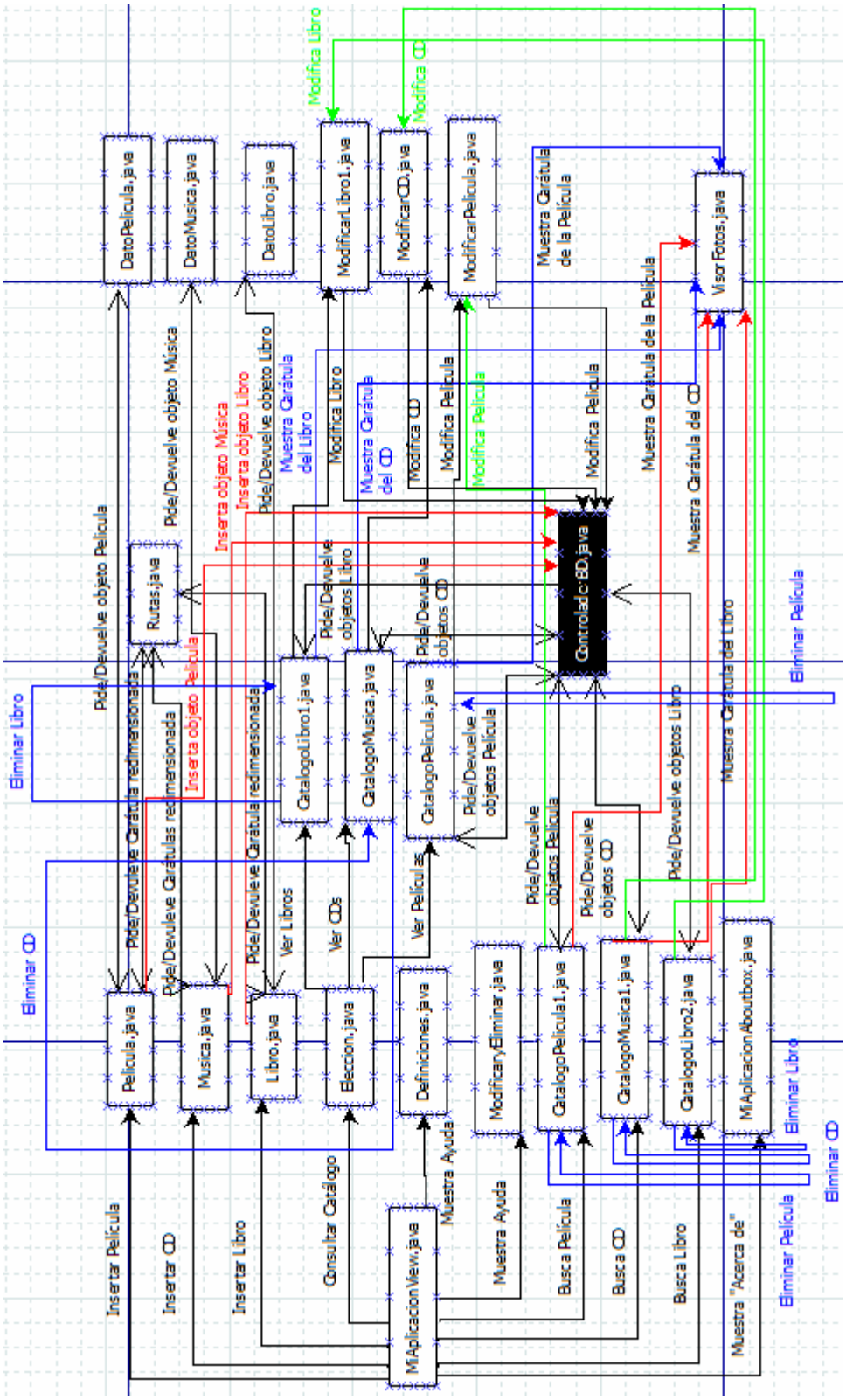

Figura 15. Diagrama de alto nivel donde se relacionan todas las clases del sistema

3.3 Diagrama E-R

En primer lugar, se realizará el análisis del universo del discurso, según el cual se recogen las entidades básicas y las relaciones que van a ser necesarias para el correcto funcionamiento de la base de datos. Como entidades básicas que tenemos que tener de acuerdo con los requisitos, se distinguen unas figuras básicas que son:

- **Administrador:** Es el encargado de mantener el correcto funcionamiento de la base de datos. A su vez tiene más tareas encomendadas, que se describirán a continuación. El administrador viene por defecto.
- **Usuario:** Es el que hace uso del programa y en el caso de nuestra aplicación el usuario será local y al ser una aplicación local no necesitara identificarse como si fuera una aplicación Web ya que de la aplicación local solo hace uso su propietario.

Una vez definidas estas dos primeras entidades, es necesaria una entidad que englobe a los objetos de los diferentes tipos que van a ser catalogados. Esto hay que hacerlo de tal manera, que sea actualizable, ya que a medida que la aplicación sea usada, se podrán introducir nuevos tipos de objeto, y estas actualizaciones no pueden variar el esquema de la base de datos de manera sustancial. Cuanto menor sea el cambio a ejercer en la base de datos, más rápido será introducido el cambio.

A su vez, el sistema, aunque no se espera que tenga una gran cantidad de objetos por ser destinado a uso doméstico, por mayor rapidez es conveniente separar de alguna manera los objetos, ya que van a tener diferentes atributos e incluso algún objeto más que otros, para poder acelerar las búsquedas y diferenciar los objetos de alguna manera. También se podría hacer de manera global, todo relacionado, pero que los atributos se encuentre la diferencia entre ellos.

Se puede pensar en un súper-tipo objeto, y que de él deriven varios subtipos que serán los elementos a catalogar, diferenciados por su categoría o un campo discriminante.

El administrador viene por defecto ya que el usuario será el mismo administrador de su aplicación. No obstante, no podrá crear nuevas categorías puesto que si se deja esta labor en manos del usuario pueden llegar a existir muchas categorías e incluso dos o más sobre el mismo tema. Como esto tiene que ser controlado, el perfil del administrador es más adecuado para realizar esta tarea. Así el usuario tendrá disponibles las categorías que ha creado el administrador por defecto.

Las categorías están compuestas de objetos. Con lo que existe una relación entre categoría existente y objeto. Todo objeto tiene que estar en una categoría. De esta forma, queda toda la información enlazada entre la categoría, el objeto y el usuario.

En resumen, las entidades y relaciones que se identifican son las siguientes:

RELACIONES

- CREA: Existe una relación crea entre el administrador y la categoría.
- TIENE: Existe una relación entre la categoría creada y el objeto al que pertenece.
- TIENE: Existe una relación entre el objeto de la categoría y el usuario para formar su catalogo.

- PRESTA: Existe una relación entre el usuario y el objeto para indicar el préstamo de dicho objeto.
- GENERALIZACIÓN: Indica de qué tipo es el objeto creado.

ENTIDADES

- ADMINISTRADOR: El administrador debe estar identificado mediante sus datos personales, uno de ellos que lo identifique del resto de administradores que existan, más un password para entrar en la aplicación, además de un correo electrónico.
- USUARIO: EL usuario estará identificado de manera similar al administrador. Se pedirán sus datos personales, más un alias. Este alias, será único por cada usuario. También se tiene que recoger un correo electrónico, que nos servirá para el alta de dicho usuario en la aplicación, como para informar de los cambios que existan en la aplicación. Así tenemos un contacto entre usuarios y administradores.
- OBJETO: El objeto será una entidad abstracta. Tiene por misión englobar a todos los objetos de todas las categorías, tanto predefinidas en la aplicación, como categorías futuras.
    - SUBTIPOS: Cada subtipo heredará los atributos que tenga el objeto, más los suyos propios, que no tienen por qué ser iguales en todos los subtipos.
- CATEGORIA: Su creación como se ha comentado está supeditada al administrador. Solo él/ellos pueden crear las categorías. Como las categorías tienen objetos, será a partir de ese momento cuando se puedan crear objetos de la categoría nueva. Un objeto siempre pertenece a la categoría.

La Figura 16 muestra el diagrama entidad-relación resultado.

La descripción completa de la base de datos se encuentra en el Anexo 1.

3.4 Descripción de clases

En este apartado se detallan todas las clases más importantes desarrolladas para **CLOC**. Para cada clase, se explican sus atributos y métodos fundamentales.

Clase ControladorBD.java

Esta clase es la más importante de todo el proyecto ya que aparte de manejar toda la base de datos es la clase que mas interactúa con el resto de las clases, tanto para guardar datos en la base de datos como para dárselos a las clases java que los piden. A continuación se describen sus atributos y métodos y también el constructor de esta clase ya que en él se realiza la conexión a la base de datos que siempre se hace antes de querer realizar cualquier operación sobre dicha base.

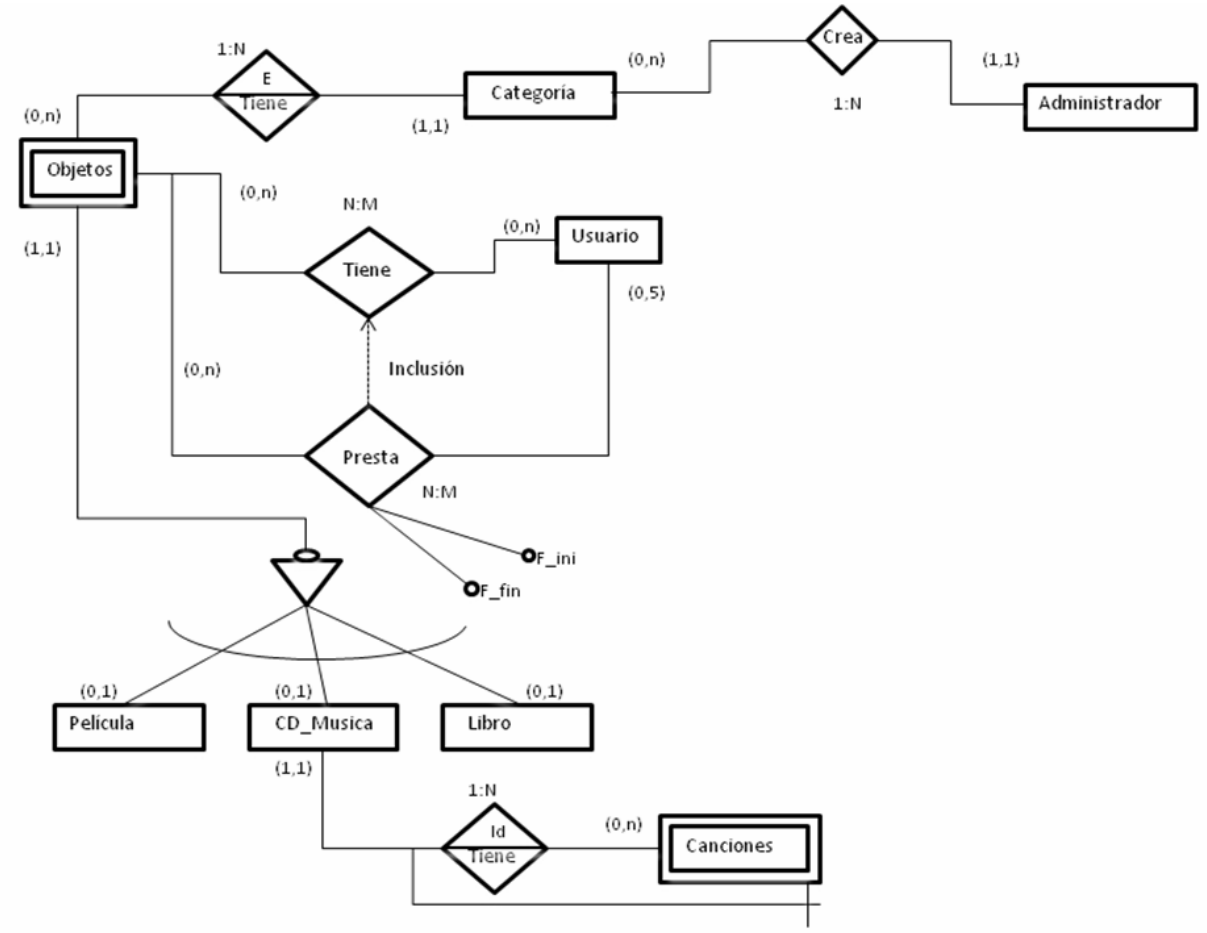

Figura 16. Diagrama E-R

Atributos:

Los atributos son cadenas de caracteres para guardar respectivamente el nombre de la base de datos, el usuario MySQL, la clave, la ruta de la base de datos y una variable para realizar la conexión a la BD.

static String **bd = "prueba";** : nombre de la base de datos.

static String **login = "root";** : usuario por defecto de MYSQL.

static String **password ="47498125";** : password de MYSQL.

static String **url = "jdbc:mysql://localhost/"+bd;** : localización de la base de datos.

public Connection **cadenaConexion = null;** : variable para realizar la conexión.

Constructor:

A continuación se muestra las instrucciones importantes que realiza el constructor de esta clase ya que son instrucciones muy importantes que hay que realizar siempre que se quiera manejar la BD, que básicamente se refiere a la conexión a la BD.

Class.forName("com.mysql.jdbc.Driver").newInstance();

Con esta instrucción se instancia el driver.

cadenaConexion = DriverManager.getConnection(url,login,password);

Con esta instrucción se realiza la conexión.

Métodos:

public void **InsertarAdminDefecto()**: Inserta un administrador por defecto, siempre que no este en la BD, con unos valores fijos, ya que como es una aplicación local no va a haber nada más que el usuario local de la aplicación (a diferencia de un sistema Web en el que podría haber varios administradores y usuarios).

public void **InsertMusiDefecto()**: Inserta la categoría Música por defecto, siempre y cuando no esté ya en la BD.

public void **InsertPeliDefecto()**: Inserta la categoría Película por defecto, siempre y cuando no esté ya en la BD.

public void **InsertLibDefecto()**: Inserta la categoría Película por defecto, siempre y cuando no esté ya en la BD.

public int **SiguienteID_Objeto()**: Obtiene el siguiente id_objeto para cuando se inserte un nuevo objeto en la tabla objeto sepa cual es el identificador que le corresponde.

public int **InsertarObjeto(int categoria)**: Inserta en la tabla objeto un objeto(Libro, Película o CD) con su correspondiente categoría.

public int **ObtenerIDCategoria(String nombrecat)**: Método que recibe un nombre de categoría y obtiene su correspondiente ID de categoría.

public void **InsertarMusica(DatoMusica objmusi)**: Método que recibe un objeto de tipo música y lo inserta en la tabla objeto y en la tabla cd_musica.

public void **InsertarPelicula(DatoPelicula objpel)**: Método que recibe un objeto de tipo película y lo inserta en la tabla objeto y en la tabla película.

public void **InsertarLibro(DatoLibro objlib)**: Método que recibe un objeto de tipo libro y lo inserta en la tabla objeto y en la tabla libro.

public boolean **ExisteCategoria(String s)**: Método que dice si existe una categoría o no en la tabla categoría según el String pasado en la llamada.

public ArrayList<DatoMusica> **GetDatosMusica()**: Método que devuelve un array con todos los objetos de tipo música que hay en la BD.

public ArrayList<DatoLibro> **GetDatosLibro()**: Método que devuelve un array con todos los objetos de tipo libro que hay en la BD.

public ArrayList<DatoPelicula> **GetDatosPelicula()**: Método que devuelve un array con todos los objetos de tipo película que hay en la BD.

public ArrayList<DatoLibro> **GetLibroEspecifico(String titu)**: Método que devuelve un array con todos los objetos de tipo libro que hay con el nombre pasado en la llamada en la BD, este array puede ser vacío ya que puede que no haya ningún objeto con el nombre pasado.

public ArrayList<DatoPelicula> **GetPeliculaEspecifica(String nomb)**: Método que devuelve un array con todos los objetos de tipo película que hay con el nombre pasado en la llamada en la BD, este array puede ser vacío ya que puede que no haya ningún objeto con el nombre pasado.

public ArrayList<DatoMusica> **GetMusicaEspecifica(String tit)**: Método que devuelve un array con todos los objetos de tipo música que hay con el nombre pasado en la llamada en la BD, este array puede ser vacío ya que puede que no haya ningún objeto con el nombre pasado.

public void **BorrarMusica(int iden_obj)**: Método que borra un objeto música de la BD según el id _ objeto que identifica ese objeto.

public void **BorrarLibro(int iden_obj)**: Método que borra un objeto libro de la BD según el id _ objeto que identifica ese objeto.

public void **BorrarPelicula(int iden_obj)**: Método que borra un objeto película de la BD según el id _ objeto que identifica ese objeto.

public void **ModificarMusica(String modi, String valu, int iden_obj)**: Método que modifica un atributo cuyo nombre será *modi*, su valor *valu* y el identificador del objeto que va a modificar *iden_obj*, de la tabla cd_musica (todo atributo excepto las dos carátulas que se gestionan de forma distinta).

public void **ModificarPelicula(String modi, String valu, int iden_obj)**: Método que modifica un atributo cuyo nombre será *modi*, su valor *valu* y el identificador del objeto que va a modificar *iden_ob* , de la tabla película (todo atributo excepto las carátula que se gestiona de forma distinta).

public void **ModificarLibro(String modi, String valu, int iden_obj)**: Método que modifica un atributo cuyo nombre será *modi*, su valor *valu* y el identificador del objeto que va a modificar *iden_obj*, de la tabla libro (todo atributo excepto las carátula que se gestiona de forma distinta).

public Image **ObtenerCaratula(String eleccion, String tabla, int ID)**: Método que devuelve la carátula de la tabla *tabla* del objeto con el id_objeto *ID* y que tiene un campo elección que se utiliza para el caso que quieras obtener la carátula frontal o trasera de un CD y poder diferenciarlas.

public void **ModificarCaratulaMusica(BufferedImage img, int ID, String tipo)**: Método que modifica la carátula de un CD donde *img* es la nueva imagen a guardar, *ID* es el identificador de objeto del CD y tipo para diferenciar si se modifica la carátula frontal o trasera.

public void **ModificarCaratulaPelicula(BufferedImage img, int ID)**: Método que modifica la carátula de una película donde *img* es la nueva imagen a guardar e *ID* es el identificador de objeto de la película.

public void **ModificarCaratulaLibro(BufferedImage img, int ID)** : Método que modifica la carátula de un libro donde *img* es la nueva imagen a guardar e *ID* es el identificador de objeto del libro.

Clase DatoLibro.java

Esta clase se utiliza para construir objetos de tipo libro. Es una clase intermedia que recoge los valores que se introducen por el interfaz y construye a partir de ellos un objeto libro con esos valores ya que ese objeto de tipo libro lo recogerá la clase **ControladorBD.java** e insertará ese objeto en la base de datos. También se utiliza para construir objetos recogiendo la información de la base de datos y luego poder usarlos. A continuación, se describen los atributos y métodos que pertenecen a esta clase.

Atributos:

private String **tituloL:** Atributo que se utiliza para guardar el titulo de un libro.

private String **autorL:** Atributo que se utiliza para guardar el autor de un libro.

private String **identificadorL:** Atributo que se utiliza para guardar el identificador(ISBN) de un libro.

private int **edicionL:** Atributo que se utiliza para guardar la edición de un libro.

private BufferedImage **caratulaL:** Atributo que se utiliza para guardar la carátula de un libro.

private String **sinopsisL**: Atributo que se utiliza para guardar la sipnosis de un libro.

private int **IDLibro:** Atributo que se utiliza para guardar el Id_Objeto de un libro. Este atributo es utilizado internamente en la base de datos para identificar únicamente a cada libro.

Métodos:

public String **GetTitulo():** Método que devuelve el titulo de un objeto libro.

public void **SetTitulo(String tit):** Método que establece el titulo *tit* de un objeto libro.

public String **GetAutor()**: Método que devuelve el autor de un objeto libro.

public void **SetAutor(String aut)**: Método que establece el autor *aut* de un objeto libro.

public String **GetIdentificador()**: Método que devuelve el identificador(ISBN) de un objeto libro.

public void **SetIdentificador(String iden):** Método que establece el identificador(ISBN) *iden* de un objeto libro.

public int **GetEdicion():** Método que devuelve la edición de un objeto libro.

public void **SetEdicion(int edi):** Método que establece la edición *edi* de un objeto libro.

public BufferedImage **GetCaratula()**: Método que devuelve la carátula de un objeto libro.

public void **SetCaratula(BufferedImage car):** Método que establece la carátula *car* de un objeto libro.

public String **GetSinopsis():** Método que devuelve la sinopsis de un objeto libro.

public void **SetSinopsis(String sinop):** Método que establece la sinopsis *sinop* de un objeto libro.

public void **SetIDLibro(int obj):** Método que establece el id_objeto *obj* de un objeto libro que identifica únicamente a este objeto en la BD.

public int **GetIDLibro():** Método que devuelve el id_objeto de un objeto libro que identifica únicamente a este objeto en la BD.

Clase DatoMusica.java

Esta clase se utiliza para construir objetos de tipo música. Es una clase intermedia que recoge los valores que se introducen por el interfaz y construye a partir de ellos un objeto música con esos valores ya que ese objeto de tipo música lo recogerá la clase **ControladorBD.java** e insertará ese objeto en la base de datos. También se utiliza para construir objetos recogiendo la información de la base de datos y luego poder usarlos. A continuación, se describen los atributos y métodos que pertenecen a esta clase.

Atributos:

private String **tituloM**: Atributo que se utiliza para guardar el título de un CD de música.

private String **autorM**: Atributo que se utiliza para guardar el autor de un CD de música.

private int **ndiscosM**: Atributo que se utiliza para guardar el número de discos de un CD de música.

private BufferedImage **caratulafM**: Atributo que se utiliza para guardar la carátula frontal de un CD de música.

private BufferedImage **caratulatM**: Atributo que se utiliza para guardar la carátula trasera de un CD de música.

private String **generoM**: Atributo que se utiliza para guardar el género de un CD de música.

private int **IDMusi**: Atributo que se utiliza para guardar el Id_Objeto de un CD. Este atributo es utilizado internamente en la base de datos para identificar únicamente a cada CD de música.

Métodos:

public String **GetTituloM()**: Método que devuelve el título de un objeto música.

public void **SetTituloM(String tit)**: Método que establece el título *tit* de objeto música.

public String **GetAutorM()**: Función que devuelve el autor de un objeto música.

public void **SetAutorM(String aut)**: Método que establece el autor *aut* de un objeto música.

public int **GetNdiscos()**: Método que devuelve el número de discos de un objeto música.

public void **SetNdiscos(int disc)**: Método que establece el número de discos *disc* de un objeto música.

public BufferedImage **GetCaratulaF()**: Método que devuelve la carátula frontal de de un objeto música.

public void **SetCaratulaF(BufferedImage car)**: Método que establece la carátula frontal *car* de un objeto música.

public BufferedImage **GetCaratulaT()**: Método que devuelve la carátula trasera de un objeto música.

public void **SetCaratulaT(BufferedImage car)**: Método que establece la carátula trasera *car* de un objeto música.

public String **GetGeneroM()**:Método que devuelve el género de un objeto música.

public void **SetGeneroM(String gen)**: Método que establece el género *gen* de un objeto música.

public int **GetIDMusica()**:Método que devuelve el id_objeto de un objeto música que identifica únicamente a este objeto en la BD.

public void **SetIDMusica(int obj)**: Método que establece el id_objeto *obj* de un objeto música que identifica únicamente a este objeto en la BD.

Clase DatoPelicula.java

Esta clase se utiliza para construir objetos de tipo película. Es una clase intermedia que recoge los valores que se introducen por el interfaz y construye a partir de ellos un objeto película con esos valores ya que ese objeto de tipo película lo recogerá la clase **ControladorBD.java** e insertará ese objeto en la base de datos. También se utiliza para construir objetos recogiendo la información de la base de datos y luego poder usarlos. A continuación, se describen los atributos y métodos que pertenecen a esta clase.

Atributos:

private String **nombreP**: Atributo que se utiliza para guardar el nombre de una película.

private int **soporteP**: Atributo que se utiliza para guardar el soporte de una película.

private BufferedImage **caratulaP**: Atributo que se utiliza para guardar la carátula de una película.

private String **formatoP**: Atributo que se utiliza para guardar el formato de una película.

private String **directorP**: Atributo que se utiliza para guardar el director de una película.

private int **IDPeli** Atributo que se utiliza para guardar el Id_Objeto de una película. Este atributo es utilizado internamente en la base de datos para identificar únicamente a cada película.

Métodos:

public String **GetNombreP()**: Método que devuelve el nombre de un objeto película.

public void **SetNombreP(String nom)**: Método que establece el nombre *nom* de objeto película.

public int **GetSoporte()**: Método que devuelve el soporte de un objeto película.

public void **SetSoporte(int sop)**: Método que establece el soporte *sop* de objeto película.

public BufferedImage **GetCaratulaP()**: Método que devuelve la carátula de un objeto película.

public void **SetCaratulaP(BufferedImage car)**: Método que establece la carátula *car* de un objeto película.

public String **GetFormato()**: Método que devuelve el formato de un objeto película.

public void **SetFormato(String form)**: Método que establece el formato *form* de un objeto película.

public String **GetDirector()**: Método que devuelve el director de un objeto película.

public void **SetDirector(String direc)**: Método que establece el director *direc* de un objeto película.

public int **GetIDPelicula()**: Método que devuelve el id_objeto de un objeto película que identifica únicamente a este objeto en la BD.

public void **SetIDPelicula(int obj)**: Método que establece el id_objeto *obj* de un objeto película que identifica únicamente a este objeto en la BD.

Clase Rutas.java

Esta clase, junto con la clase **ControladorBD** es otra de las más importantes del proyecto ya que se encarga de dos de las funciones más importantes y complejas del proyecto en si. Estas dos funciones son **ejecutarBotonFichero** y **loadImageForScreen,** y se encargan de abrir el selector de ficheros seleccionando la imagen correspondiente y redimensionar la imagen en caso de que sea necesario respectivamente. A continuación, se verán todos lo métodos que posee esta clase ya que no dispone de atributos como las otras clases que se han visto anteriormente.

Métodos:

public void **cambiarImagen(File file, javax.swing.JTextField rellena)**: Método que se utiliza para devolver en el campo *rellena,* pasado por referencia, la ruta path del fichero *file* pasado en la cabecera de la función. Consigue la ruta absoluta del fichero y la escribe en el elemento rellena para que este consiga la localización del fichero, eso siempre que desde esa ruta se pueda cargar correctamente el fichero imagen, ya que sino nos mostraría un mensaje por pantalla debido a que se puede cargar la imagen.

public void **ejecutarBotonFichero(java.awt.event.ActionEvent evt, javax.swing.JTextField rellenar) :** Método que se utiliza para abrir el selector de ficheros de imágenes seleccionando la imagen que se desee , que a su vez solo va a permitir que se puedan seleccionar archivos de tipo imagen como *.jpeg, .giff, jpg, tiff, png*…, y una vez realizados estos pasos esta porpia funciona llamara a su vez a **cambiarImagen** que devolvera en el campo rellenar por referencia la ruta absoluta(path) del fichero seleccionado en la función **ejecutarBontonFichero**.

public static BufferedImage **loadImageForScreen(String fileName) throws ParameterException**: Método que se ocupa de que si la imagen es demasiado grande para mostrarla entera por pantalla, superando por tanto unos valores máximos constantes fijados, se redimensione a un tamaño adecuado a la pantalla según las dimensiones proporcionales de esta. La imagen se redimensiona, si es necesario, antes de que se guarde en la base de datos para luego simplemente recuperarla y mostrarla directamente.

public static byte[] **imageToByteArray(BufferedImage image) throws IOException**: Método que utiliza principalmente la clase **ControladorBD.java** a la hora de tener que insertar la carátula de un CD, Libro o Película ya que lo que se ocupa es de pasar un objeto de tipo *Image* a un array de *bytes* ya que así resulta más eficiente.

Resto de clases

En este apartado se describen conjuntamente el resto de las clases ya que estas clases tienen una funcionalidad más de diseño de interfaz, programación por eventos y no dispone de una serie de atributos y funciones para modificar/devolver sus valores como las clases anteriores.

En este tipo de clases abundan los elementos para diseñar elementos del interfaz como marcos, botones de aceptar, visor de imágenes etc. y sobre todo la programación guiada por eventos que es la más utilizada en este tipo de clases. Este tipo de programación consiste en asociar una acción a un elemento del interfaz, como por ejemplo, el típico botón examinar que al hacer clic sobre él, nos muestra un selector de ficheros. La acción en este ejemplo es hacer clic sobre el botón y que se abra el selector de archivos. Una vez dicho esto, se van a explicar las funciones principales de las que se encargan el resto de las clases que faltan:

MiAplicacionView.java, MiAplicacionApp.java, MiapliacionAboutBox.java:
Son tres clases que se encargan de la apariencia inicial del sistema, de la función *main* que se encarga de iniciar la aplicación y de mostrar información "Acerca de la aplicación" respectivamente por cada clase.

CatalogoLibro1.java, CatalogoMusica.java, CatalogoPelicula.java: Estas tres clases son las encargadas de diseñar cada uno de los diferentes catálogos, donde se muestran los diferentes títulos, para películas, CDs y libros y que posee funciones para eliminar, modificar, o ver las carátulas de cada uno de los elementos mostrados en su catalogo correspondiente.

CatalogoLibro2.java, CatalogoMusica1.java, CatalogoPelicula1.java: Realizan las misma función que las clases anteriores pero la única diferencia es que esta clases son utilizadas para mostrar elementos específicos pedidos por el buscador utilizado en CLOC a diferencia de que las clases anteriores muestran todos los elementos de su categoría respectiva.

Musica.java, Película.java, Libro.java: Son clases que se encargan de diseñar los formularios para insertar Libros, Películas y CDS y una vez rellenados por el usuario utilizan las clases *DatoLibro.java, DatoPelicula.java, DatoMusica.java y ControladorBD.java* para construir un objeto de cada tipo respectivo y insertarlo en la BD.

ModificarCD.java, ModificarLibro1 y ModificarPelicula.java: Son clases que se utilizan para diseñar el interfaz de cuando se quiere modificar un campo de un CD, Libro o Película. Además, utilizan la clase *ControladorBD.java* para poder modificar el campo del objeto seleccionado y actualizarlo en la BD.

Definiciones.java y ModificaryEliminar.java: Son clases que se utilizan en le menú ayuda para ayudar al usuario a entender algunos términos en caso de duda.

Eleccion.java: es una clase que muestra un menú de elección para que el usuario seleccione el catálogo que desea ver.

4. Pruebas y resultados

Las pruebas se han realizado en dos partes claramente diferenciadas. En primer lugar se ha comprobado el correcto funcionamiento que el programa, y en segundo lugar se ha verificado que el programa cumple los requisitos que se definieron para el proyecto.

Se han realizado las siguientes pruebas:

1. Pruebas unitarias
2. Pruebas de integración
3. Pruebas de validación
4. Pruebas de aceptación
5. Verificación de requisitos

La estructura de las pruebas se ha realizado siguiendo un enfoque ascendente desde las pruebas unitarias de cada módulo de la aplicación hasta valorar si la aplicación cumple con los requisitos no funcionales, tales como que sea manejable y usable, y la comprobación del nivel de satisfacción alcanzado por usuarios con y sin conocimientos informáticos.

4.1 Pruebas unitarias

Estas pruebas se centran en comprobar en el correcto funcionamiento de cada módulo de CLOC de forma independiente. Se realizará una prueba de caja blanca sobre el algoritmo de redimensionado de imágenes debido a su mayor complejidad, y pruebas de caja negra para el resto de las clases.

4.1.1 Caja Blanca

La prueba de caja blanca se realiza sobre la clase más compleja que es **Rutas.java** y que se encarga del algoritmo de redimensionado de imágenes. A continuación, se verá un ejemplo de cómo insertar un libro en la base de datos, donde interviene **Rutas.java** y el papel que desempeña en el sistema.

Bloque 1

1. En este primer bloque se visualiza el formulario por pantalla para rellenar los campos del Libro que vamos a Insertar.

Bloque 2

2. Completar el formulario.

Bloque 3

3. Se produce el evento del botón examinar con el que muestra por pantalla el selector de ficheros y elige la imagen correspondiente siempre que se pueda cargar.

Bloque 4

4. Se produce el evento del botón Aceptar. Si se introducen valores incorrectos o se dejan campos en blanco saltará un mensaje informándole del error y le pedirá que introduzca valores correctos.

Bloque 5

5. Se comprueba que la carátula cumple con los valores máximos constantes y sino es así se renderiza la imagen utilizando la función **loadImageForScreen** de la clase **Rutas.java** y es aquí cuando se aplica el algoritmo para renderizar la imagen a un tamaño adecuado a la pantalla.

Bloque 6

6. Una vez renderizada o no la imagen se procede a construir el objeto libro utilizando las funciones de la clase *DatoLibro.java* para construir un objeto libro.

Bloque 7

7. Una vez construido ya el objeto libro se llama a la función **IsertarLibro** que recibe un objeto de tipo libro y lo inserta con esta función en la base de datos.

Una vez definidos los bloques de cómo se inserta un libro y como interactúa el algoritmo de redimensionado en esta inserción, la Figura 17 muestra el diagrama de caja blanca creado.

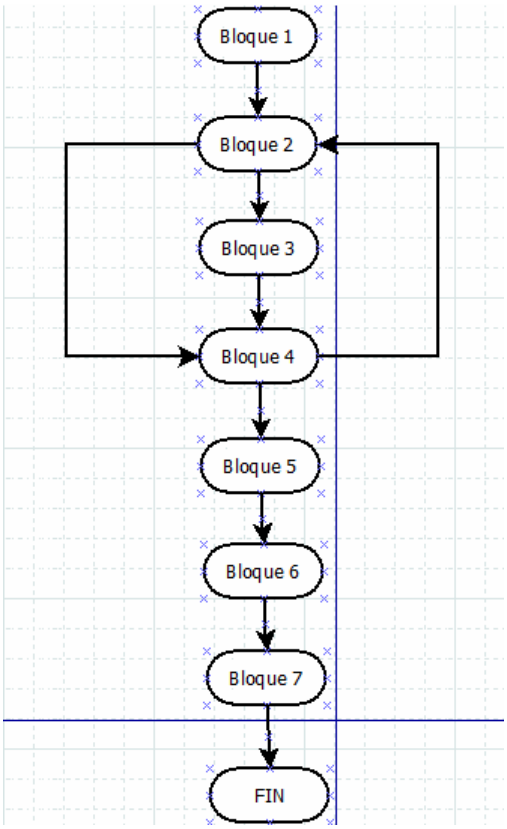

Figura 17. Diagrama de caja blanca

Como se puede observar en el diagrama, el flujo de datos se cumple para todos los distintos caminos, como por ejemplo en el caso de que los datos introducidos en el bloque 4 no sean correctos y tengamos que volver al bloque 2 para rellenar bien el formulario y así poder llegar al último bloque.

4.1.2 Caja Negra

En el apartado de pruebas de caja negra se prueban el resto de las clases con una serie de **valores límite** para comprobar si cumple o no con los requisitos no funcionales. Estos valores limite son valores que hacen llevar al limite a la aplicación para que puedan dar lugar a fallos y su correspondiente corrección. A continuación, se muestran una serie de tablas con las pruebas realizadas de caja negra.

Prueba 1

Clases java	Entrada	Descripción	Salida esperada	Salida real
M**usica.java, Película.java y Libro.java**	Todos los campos a vacío	Intenta insertar un CD, Libro o Película con todos los campos vacío	Debería mostrar un mensaje por pantalla que le obligue a rellenar los campos y no insertar en la BD	Lo inserta en la BD y por tanto el resultado es **incorrecto**

Tabla 11. Prueba 1 caja negra

Este error se corrigió en las tres clases realizando la comprobación de no insertar los datos en la base datos si hay algún campo vacío en el formulario. En su lugar, se debe informar al usuario de que debe rellenarlos.

Prueba 2

Clases java	Entrada	Descripción	Salida esperada	Salida real
Musica.java, Película.java y Libro.java	Todos los campos rellenos con caracteres e incluso los campos de tipo integer	Intenta insertar un CD, Libro o Película con todos los campos rellenos y en los campos con valor entero introducir caracteres también.	Debería mostrar un mensaje por pantalla avisando al usuario que en esos campos enteros meta solo valores enteros y correctos.	El resultado es **correcto**

Tabla 12. Prueba 2 caja negra

Prueba 3

Clases java	Entrada	Descripción	Salida esperada	Salida real
MiAplicacion View.java	Dejar el campo del buscador a vacío	Buscar un titulo con nombre vacío en Libros, Películas o CDS de Música	Debería mostrar todos los títulos que hay en el correspondiente catalogo	El resultado es **correcto**

Tabla 13. Prueba 3 de caja negra

Prueba 4

Clases java	Entrada	Descripción	Salida esperada	Salida real
Modificar CD.java, Modificar Libro.java y Modificar Pelicula.java	Dejar el campo a rellenar a vacío	Intenta insertar un valor vacío para el atributo seleccionado a modificar	Debería mostrar un mensaje por pantalla pidiendo al usuario que complete el campo.	Inserta en la BD la actualización de ese atributo y por tanto el resultado es **incorrecto**

Tabla 14. Prueba 4 de caja negra

Este error se corrigió en las tres clases realizando la comprobación de que si el campo a modificar era vacío no se debe insertar en la base de datos. En su lugar, se debe informar al usuario de que debe introducir esa información.

Prueba 5

Clases java	Entrada	Descripción	Salida esperada	Salida real
Modificar CD.java, Modificar Libro.java y Modificar Pelicula.java	Insertar en el campo caracteres	Insertar en el campo caracteres cuando el atributo a actualizar es un número entero	Debería mostrar un mensaje por pantalla pidiendo al usuario que introduzca en el campo un valor entero y correcto	El resultado es **correcto**

Tabla 15. Prueba 5 de caja negra

Prueba 6

Clases java	Entrada	Descripción	Salida esperada	Salida real
CatalogoLibro1.java, CatalogoLibro2.java, CatalogoMusica.java, CatalogoMusica1,java, CatalogoPelicula.java y CatalogoPelicula1.java	Hacer clic con el ratón en el botón "Modificar"	Hacer clic en el botón "Modificar" sin haber seleccionado un id_objeto de un Libro, Película o CD	Debería mostrar un mensaje por pantalla en el que indique que para modificar un elemento del catálogo hay que seleccionar su correspondiente id_objeto	No muestra nada por pantalla por lo que el resultado es **incorrecto**

Tabla 16. Prueba 6 de caja negra

Este error se corrigió en las seis clases, comprobando que si el usuario hace clic directamente en el botón "Modificar" sin haber seleccionado un id_objeto, se le debe indicar que debe seleccionarlo para poder realizar la modificación.

Prueba 7

Clases java	Entrada	Descripción	Salida esperada	Salida real
CatalogoLibro1.java, CatalogoLibro2.java, CatalogoMusica.java, CatalogoMusica1,java, CatalogoPelicula.java y CatalogoPelicula1.java	Hacer clic con el ratón en el botón "Eliminar"	Hacer clic en el botón "Eliminar" sin haber seleccionado un id_objeto de un Libro, Película o CD	Debería mostrar un mensaje por pantalla en el que indique que para eliminar un elemento del catálogo hay que seleccionar su correspondiente id_objeto	No muestra nada por pantalla por lo que el resultado es **incorrecto**

Tabla 17. Prueba 7 de caja negra

Este error se corrigió en las seis clases, comprobando que si el usuario hace clic directamente en el botón "Eliminar" sin haber seleccionado un id_objeto, se le debe indicar que debe seleccionarlo para poder realizar la modificación.

Prueba 8

Clases java	Entrada	Descripción	Salida esperada	Salida real
CatalogoLibro1.java, CatalogoLibro2.java, CatalogoMusica.java, CatalogoMusica1,java, CatalogoPelicula.java y CatalogoPelicula1.java	Hacer clic con el ratón en el botón “Ver Caratula ...”	Hacer clic en el botón “Ver Caratula …” sin haber seleccionado un id_objeto de un Libro, Película o CD	Debería mostrar un mensaje por pantalla en el que indique que para ver la carátula de un elemento del catalogo hay que seleccionar su correspondiente id_objeto	No muestra nada por pantalla por lo que el resultado es **incorrecto**

Tabla 18. Prueba 8 de caja negra

Este error se corrigió en las seis clases, comprobando que si el usuario hace clic directamente en el botón “Ver carátula” sin haber seleccionado un id_objeto, se le debe indicar que debe seleccionarlo para poder realizar la modificación.

Para el resto de las clases no se pueden hacer las **pruebas con valores limite** ya que son clases que o intervienen en la realización del interfaz, o informan de algunos conceptos o simplemente sirven para interactuar con las clases ya probadas.

4.2 Pruebas de integración

En este apartado de pruebas de integración se comprobó si todos los módulos funcionaban correctamente interrelacionados entre ellos. Estas pruebas se han ido desarrollando de forma sistemática según se avanzaba en el desarrollo de CLOC, comprobando que la unión de distintos módulos funcionaba correctamente. A continuación, se muestra una secuencia de diagramas con la integración de los distintos módulos para el correcto funcionamiento a la hora de insertar Libros, CDs o Películas.

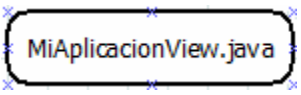

Figura 18. Prueba 1 integración

En la Figura 18, se muestra la primera clase que se creó y que contiene el interfaz principal de la aplicación. Posteriormente se añadió la clase que controla la base de datos que lo único que realizaba en ese momento era conexiones a la base de datos.

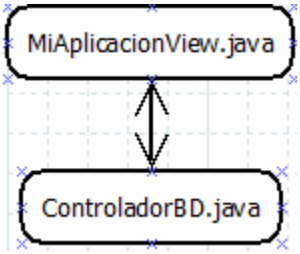

Figura 19. Prueba 2 de integración

En la Figura 19 se pueda observar la correlación entre las dos clases. A continuación, se añadieron las clases encargadas de diseñar el interfaz para insertar libros, CDs y películas como se muestra en la Figura 20.

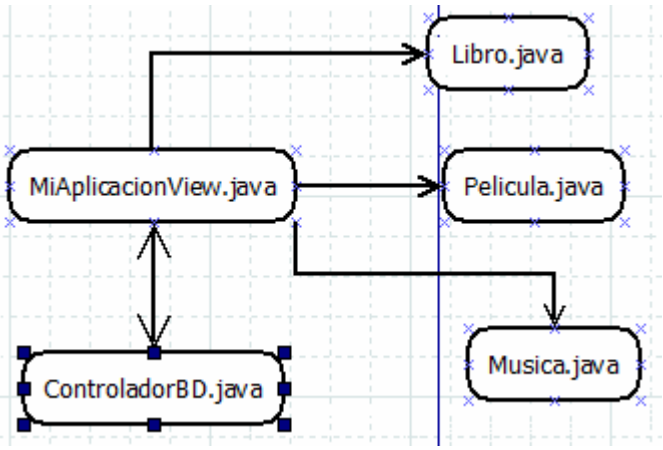

Figura 20. Prueba 3 de integración

El siguiente paso fue la integración con las clases para construir objetos de tipo libro, película y CD de música con los datos del interfaz y así una vez construido el objeto insertarlo en la BD. Con estas clases toda la estructura de la aplicación quedó organizada facilitando posibles cambios futuros o actualizaciones, como se muestra en la Figura 21.

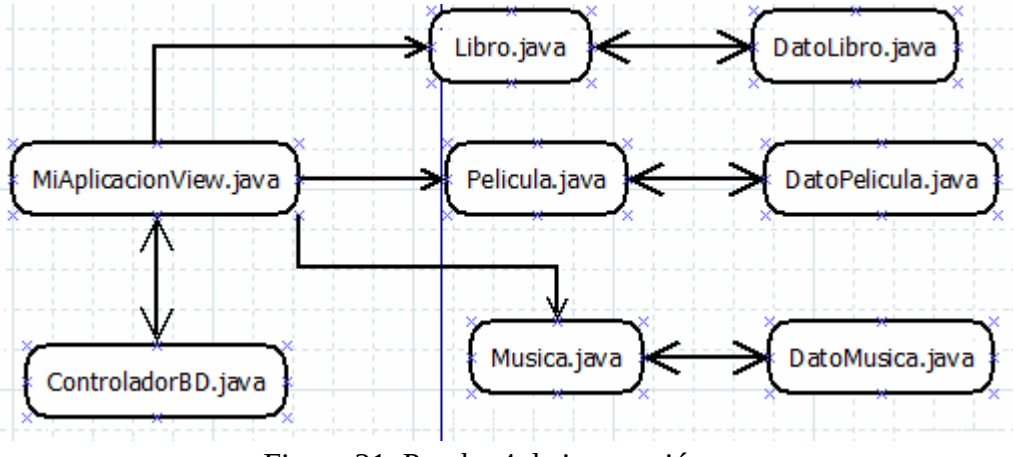

Figura 21. Prueba 4 de integración

Por ultimo, se integraron el resto de módulos de gestión de la BD como se muestra en la Figura 22.

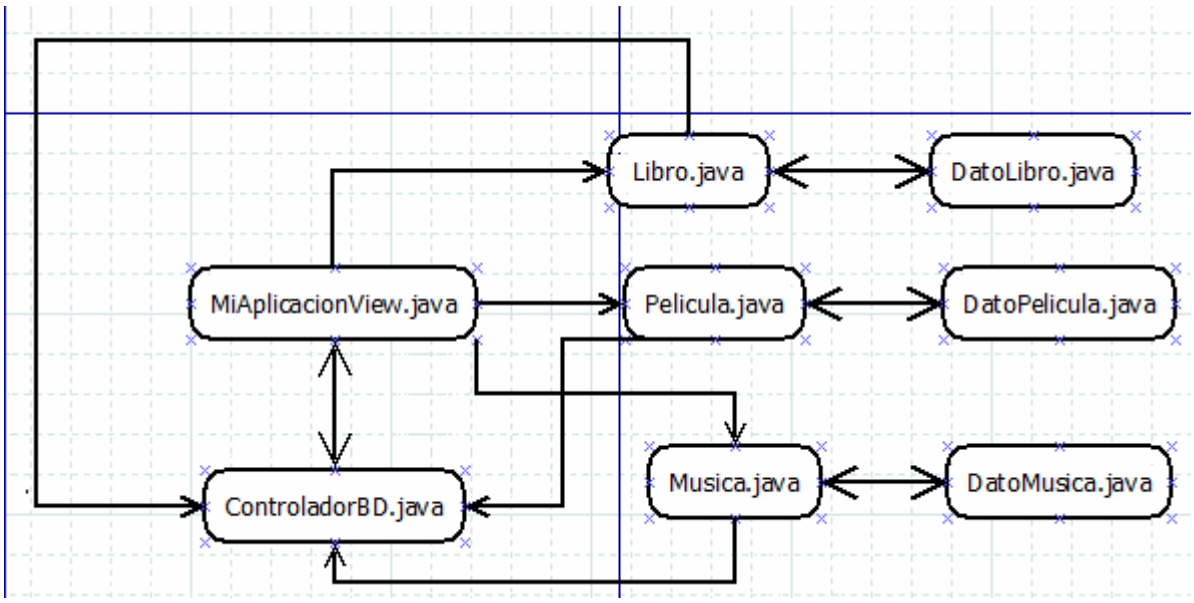

Figura 22. Prueba 5 de integración

4.3 Pruebas de validación

En estas pruebas se comprueba el correcto funcionamiento de la aplicación respecto a los requisitos descritos en la fase de especificación. Se mostrará mediante capturas de pantalla como se han ido cumpliendo estos requisitos.

Insertar un CD

Figura 23. Prueba de validación 1

Figura 24. Prueba de validación 2

En la Figura 25, se muestra que el CD se ha insertado en el catálogo de música correctamente.

Figura 25. Prueba de validación 3

Buscar un CD

Figura 26. Prueba de validación 4

En la Figura 27 se muestra el CD que se ha buscado se encuentra en el catálogo de música.

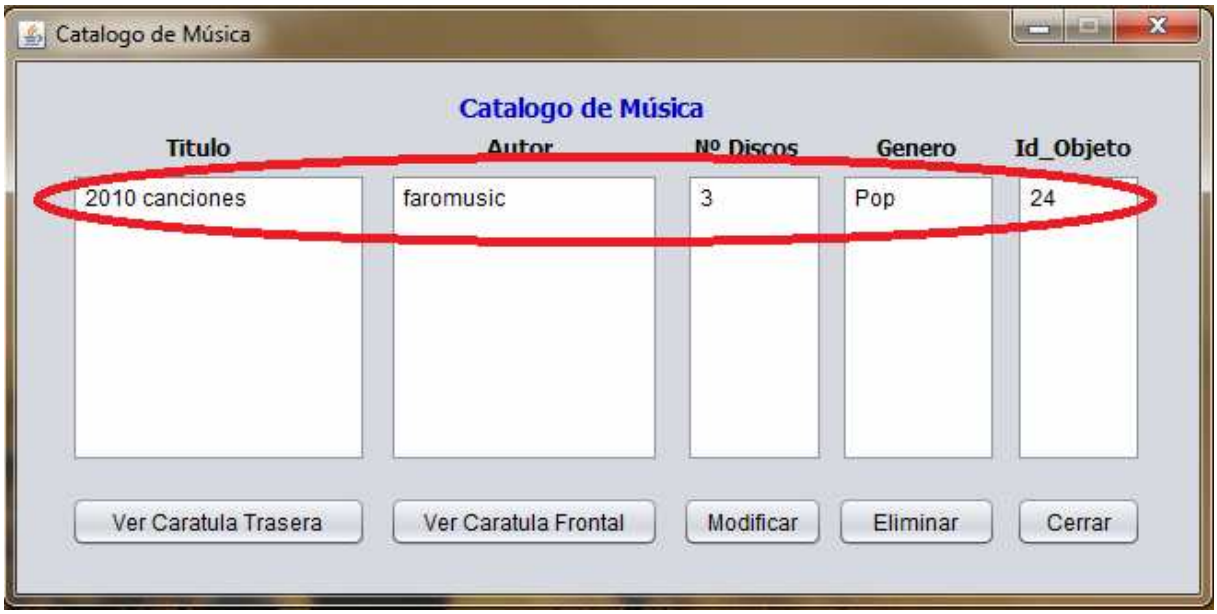

Figura 27. Prueba de validación 5

Modificar un CD

En este apartado se muestra cómo se modifica un dato de un objeto, como por ejemplo la edición de un CD.

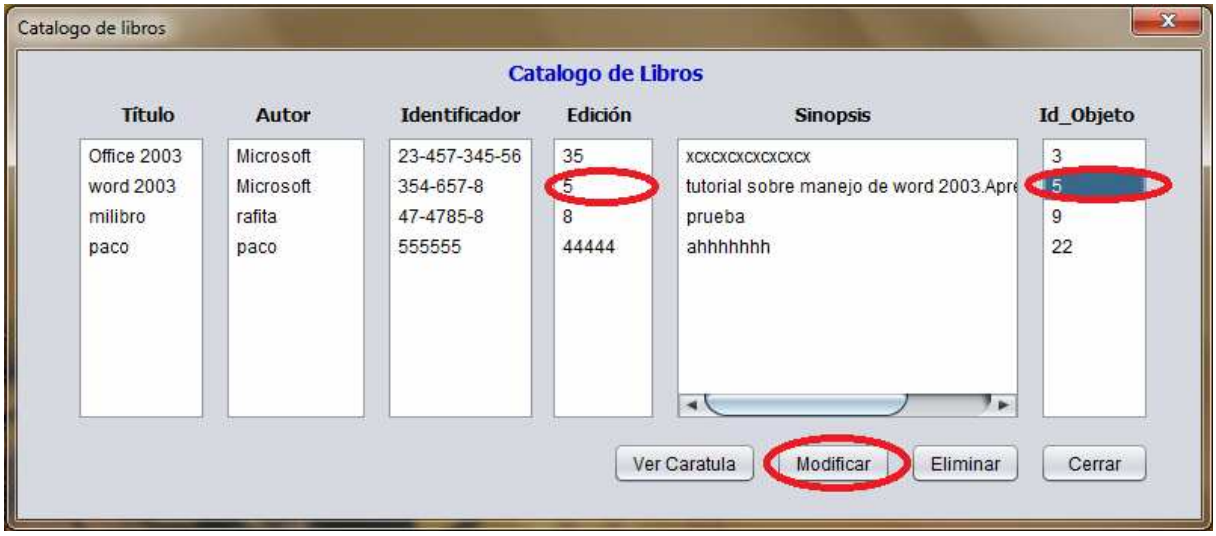

Figura 28. Prueba de validación 6

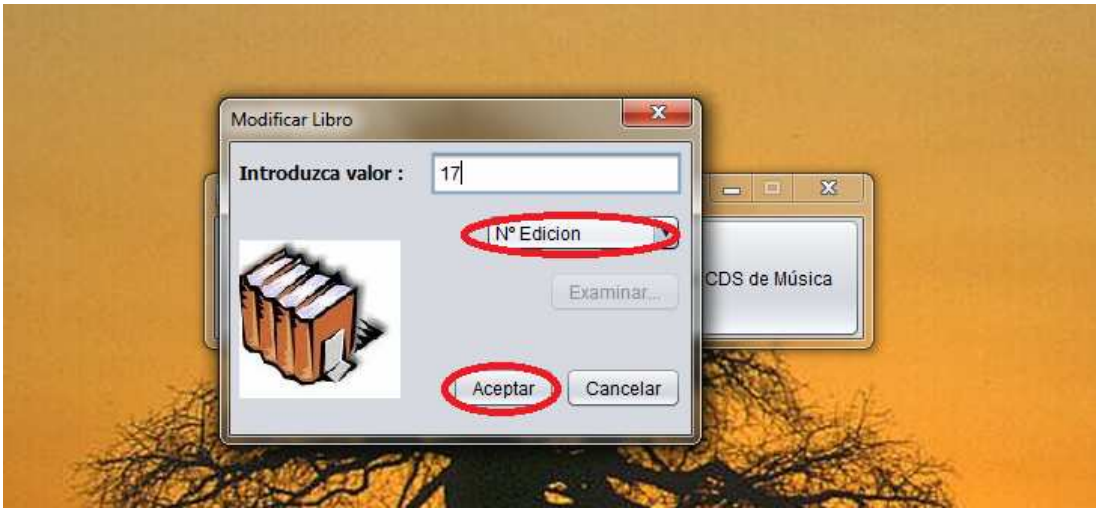

Figura 29. Prueba de validación 7

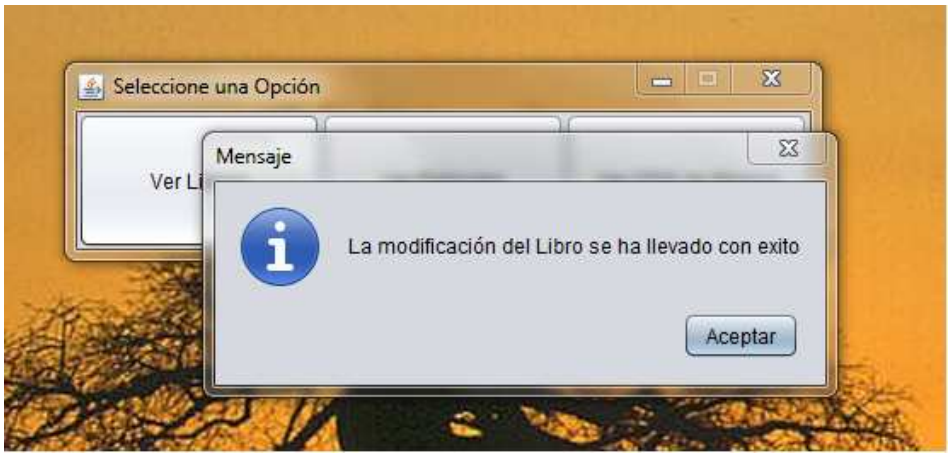

Figura 30. Prueba de validación 8

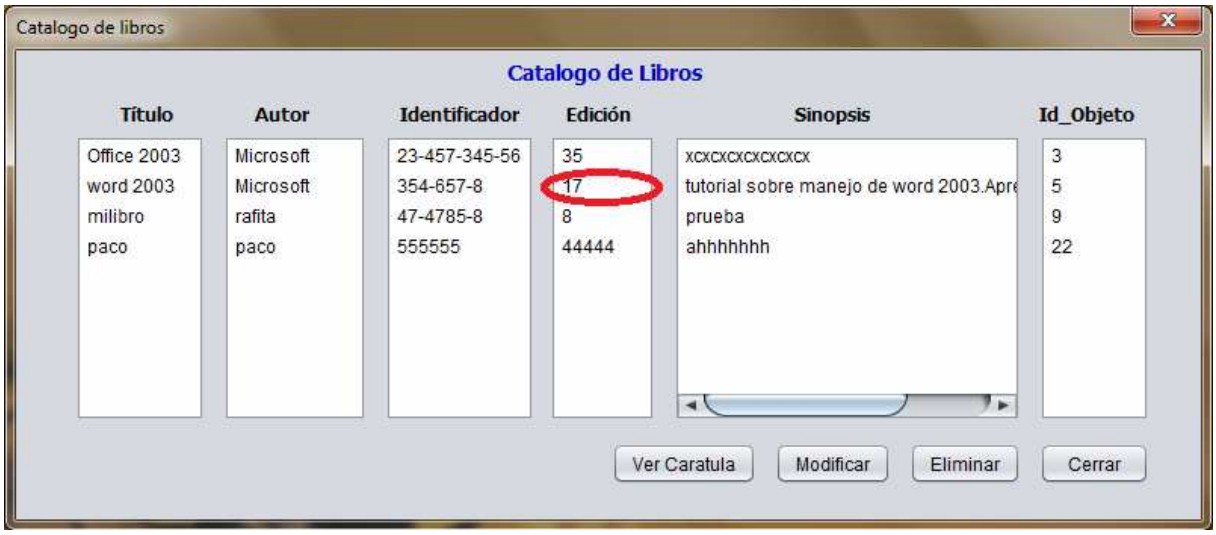

Figura 31. Prueba de validación 9

Eliminar un CD

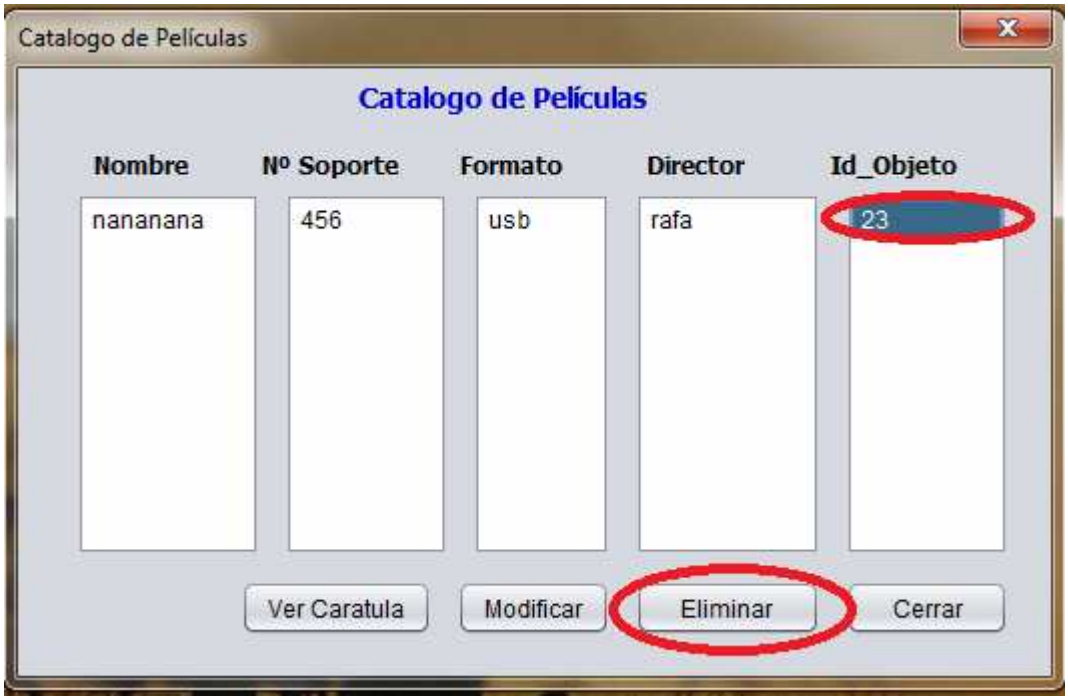

Figura 32. Prueba de validación 10

Figura 33. Prueba de validación 11

4.4 Pruebas de aceptación

Las pruebas de aceptación son las últimas que se han realizado y son las más importantes puesto que suponen la comprobación con el usuario del cumplimiento de los objetivos y requisitos enunciados en la especificación de la aplicación.

Los perfiles de los usuarios han sido elegidos aleatoriamente, pero teniendo en cuenta muy especialmente su nivel de conocimiento informático, ya que CLOC está más orientado a usuarios con conocimientos mininos de ordenadores. De esta forma, se pretende comprobar la usabilidad de la aplicación.

En total, se ha obtenido una muestra de 10 usuarios cuyas características se muestran en la Tabla 19.

	Edad	**Estudios**
U1	52	Bachillerato
U2	52	Estudios primarios
U3	82	Estudios primarios
U4	24	Formación Profesional
U5	47	COU
U6	31	Auxiliar Administrativo
U7	58	Carrera
U8	49	COU
U9	19	Universitaria
U10	19	Formación profesional

Tabla 19. Tabla de datos personales de usuarios

En primer lugar, se pidió a los usuarios que completaran siete tareas con la aplicación. Los usuarios no habían utilizado nunca antes CLOC, y sólo se les explicó durante 5 minutos brevemente sus principales características, intentando que en la medida de lo posible interactuasen de forma interactiva con la interfaz, para medir el tiempo que tardaban en completar las tareas, si había alguna que no podían completar, y en qué casos pedían más asistencia.

Estas tareas se solicitaron según el cuestionario del Anexo 2. Los diez usuarios completaron todas las tareas que se solicitaron. La Tabla 20 recoge los tiempos que dedicaron a cada una de las tareas. Es destacable como en promedio, tardan sólo 2 minutos 35 segundos en realizarlas, siendo el usuario 4 el más rápido empleando 12 segundos de media, y el más lento con 7 minutos y 8 segundos el usuario 3.

	Tarea 1	**Tarea 2**	**Tarea 3**	**Tarea 4**	**Tarea 5**	**Tarea 6**	**Tarea 7**	**M**
U1	3’	2’ 30’’	20’’	10’’	10’’	10’’	20’’	**57’’**
U2	3’30’’	4’	40’’	20’’	20’’	20’’	20’’	**1’ 21’’**
U3	15’	10’	10’	3’	3’	3’	10’	**7’ 8’’**
U4	25’’	30’’	10’’	5’’	5’’	5’’	10’’	**12’’**
U5	3’	2’	20’’	8’’	8’’	8’’	8’’	**51’’**
U6	4’	3’	20’’	10’’	10’’	10’’	15’’	**1’ 10’’**
U7	5’	4’	30’’	15’’	15’’	15’’	20’’	**1’ 31’’**
U8	2’ 34’’	1’	47’’	8’’	8’’	8’’	5’’	**41’’**
U9	2’	40’’	20’’	5’’	5’’	5’’	5’’	**28’’**
U10	2’	1’	20’’	5’’	5’’	5’’	5’’	**31’’**
M	**5’ 47’’**	**4’ 5’’**	**1’ 58’’**	**38’’**	**38’’**	**38’’**	**1’ 41’’**	**2’ 35’’**

Tabla 20. Tabla de tiempos de usuarios

Es destacable también que según los tiempos, parece que la tarea más compleja ha sido la Tarea 1 puesto que es a la que han dedicado más tiempo los usuarios (5’ 47’’ de media). Esto se podría explicar por ser la primera toma de contacto que los usuarios realizan con CLOC. Por eso, la primera tarea puede requerir más tiempo, pero según van realizando más tareas, se familiarizan con la aplicación y requieren menos tiempo en realizarlas.

En su caso, las tareas más sencillas parece ser las tareas 3, 4 y 5 puesto que son a la que han dedicado menos tiempo los usuarios (38’’ en media). Esto se podría explicar porque ya se habían familiarizado con la aplicación con la primera tarea que es la que siempre suele costar más al utilizar una aplicación nueva. También son las tareas más fáciles debido a que simplemente son tareas de consultar datos en el catalogo y no insertarlos como las tareas anteriores que requieren completar formularios con varios campos.

Es interesante también observar que la media de tiempos de los usuarios con una edad superior a 50 años oscila entre 57’’ y 7’8’’, mientras que la media de tiempos de los usuarios con una edad inferior es 12’’ y 1’10’’.

Se puede observar que la diferencia de tiempos entre las personas de menos de 50 años y los de más de 50 años no es tan grande, cumple el objetivo de usabilidad de CLOC de poder ser usado por usuarios mayores con menos conocimientos de informática.

Por último, se pidió a los usuarios que completaran voluntaria y anónimante el cuestionario que se recoge en el Anexo 3. La escala utilizada en cada caso es de 1 (mínimo-peor) a 10 (máximo-mejor). La Tabla 21 recoge los resultados de la opinión de estos usuarios respecto a varias características de CLOC.

	Facilidad de manejo	**Diseño de Interfaz**	**Rapidez**	**Opinión Personal**
Usuario 1	9	9	7	8
Usuario 2	7	8	7	7
Usuario 3	4	7	4	4.5
Usuario 4	9	9	9	9
Usuario 5	9	9	9	9
Usuario 6	7	7	7	7
Usuario 7	6.5	9	8	7.5
Usuario 8	8	8	8	8
Usuario 9	6	6	7	6.5
Usuario 10	9	8	9	8.5
MEDIA	**7.45**	**8**	**7.5**	**7.5**

Tabla 21. Tabla de evaluación de usuarios

Como se puede apreciar en la tabla, la calificación media que obtiene la facilidad de manejo es 7.45. Este valor es bastante elevado y constante en todos los usuarios, puesto que solo un usuario suspende a la aplicación. Esta persona era la primera vez que usaba el ordenador y le resultaba complicado hasta mover el ratón del ordenador, por lo que realmente constata que la aplicación requiere unos conocimientos mínimos de informática.

En cuanto al diseño de la interfaz obtiene un 8 de nota promedio. Este valor también es elevado, de hecho los usuarios comentaban que la aplicación les resultaba atractiva y muy vistosa.

El interfaz de CLOC les parecía ameno, entretenido y fácil de utilizar. Es destacable comentar aunque los usuarios decían que la imagen de fondo era bonita, les gustaría a ellos mismos poder elegirlaa su gusto. Esta parte del proyecto se deja como parte de trabajo futuro.

En cuanto a su rapidez, CLOC obtiene un 7,5 de nota media. Todos los usuarios coincidían en que la aplicación funcionaba con rapidez. La aplicación se ha probado tanto en un ordenador portátil con procesador Intel Pentium I-Core3 a 2,13GHZ con 4GB de RAM como en un Intel Pentium 4 a 3,02GHZ con 521MB de RAM y en ambos casos ha funcionado con bastate rapidez.

Por último, en general la opinión personal de los usuarios sobre CLOC es que se ha conseguido una aplicación muy fácil de utilizar (usable), amena gracias su interfaz y útil gracias a la facilidad para catalogar estos objetos. Ocho usuarios coinciden en que la utilizarían en casa para gestionar sus libros, películas y CDs; dos usuarios, sin embargo, no la utilizarían porque piensan es más útil para empresas.

5. Conclusiones y trabajo futuro

El objetivo de este proyecto es desarrollar un sistema de gestión local, llamado CLOC, de películas, CDs de música y libros que los usuarios tienen en su domicilio de forma sencilla. La aplicación está especialmente orientada a personas con conocimientos mínimos de informática, no obstante, también puede ser utilizada por usuarios expertos en el campo de la informática.

Se decidió utilizar la librería gráfica Swing de Java para facilitar la interacción con la aplicación mediante el uso de menús. Esto exigió un alto esfuerzo de aprendizaje ya que al no saber programar con Java, se tuvo que aprender desde cero tanto orientación a objetos como gestionar interfaces gráficas.

Al principio resultó muy complicado colocar correctamente todos lo elementos del interfaz y entender la programación por eventos. No obstante, el interés de saber gestionar este tipo de elementos, me ayudó a familiarizarme con esta librería y conseguir desarrollar aplicación CLOC.

El desarrollo de la aplicación ha seguido una metodología de ciclo de vida en cascada con iteración. En la fase de análisis, se establecieron los siguientes requisitos funcionales:

- **REQ1**: El usuario tendrá la posibilidad de insertar CDs de música, películas y libros.
- **REQ2:** El usuario tendrá la posibilidad de consultar tanto películas, como CDs de música como libros tenga almacenados en CLOC.
- **REQ3:** El usuario tendrá la posibilidad de buscar un libro, película o CD de música utilizando el buscador que utiliza la aplicación en lugar de consultar el catalogo directamente por lo que resultara mas sencillo encontrar un titulo.
- **REQ4:** El usuario tendrá la posibilidad de poder eliminar un libro, película o CD de música del catalogo en cualquier momento.
- **REQ5:** El usuario podrá modificar cualquier campo de un libro, película o CD de música ya sea por modificación de una equivocación al insertar el título o por actualización del mismo titulo.
- **REQ6:** El usuario tendrá la posibilidad de ver las carátulas de los CDs de música, películas y libros en cualquier momento gracias al visor de carátulas.

Todos los requisitos antes enunciados han sido validados por los usuarios satisfactoriamente.

También se establecieron los siguientes requisitos no funcionales:

- **REQ1NF**: **CLOC** contará con un sistema de interfaz ameno y usable para favorecer la comunicación con el usuario.
- **REQ2NF**: **CLOC** gestionará las acciones del usuario proporcionando un rendimiento óptimo de una forma ágil cumpliendo con las restricciones de tiempo.
- **REQ3NF**: **CLOC** podrá utilizarse desde cualquier ordenador y cualquier sistema operativo siempre que el ordenador cuente con su propia máquina virtual de Java para ejecutar la aplicación.

Estos requisitos no funcionales también se cumplieron satisfactoriamente, demostrando las pruebas con usuarios que se ha diseñado un interfaz ameno y usable por los usuarios, con un rendimiento satisfactorio y ejecutable en varios ordenadores.

Durante el análisis, se realizaron 6 casos de uso y 6 diagramas de interacción para identificar claramente la funcionalidad del programa y tener suficiente base para comenzar con el diseño de la aplicación.

Durante el diseño, se completó el diagrama E-R, y el diagrama de alto nivel con la arquitectura de alto nivel con la relación de clases.

La codificación supuso un alto esfuerzo puesto que era la primera vez que se programaba una interfaz gráfica y fue necesario estudiar los eventos. Aprendí cómo poder provocar un evento a partir de una acción del usuario y que ese evento provocado por el usuario diera como resultado otra acción satisfactoria para el usuario.

Se fueron realizando pruebas durante todo el proceso, desde las unitarias para comprobar el correcto funcionamiento de cada clase de forma aislada, hasta las de aceptación con usuarios. En particular, se ha dado especial importancia a las pruebas con usuarios puesto que son verdaderamente el objetivo para los que se desarrolla CLOC. Son ellos, los únicos que pueden realmente validar que se ha cumplido que CLOC sea usable y útil.

10 usuarios pudieron completar 7 tareas a realizar con CLOC sin haber usado previamente el sistema y con un alto grado de satisfacción (7,5, en una escala de 1-10).Por lo tanto, se puede concluir que CLOC cumple con los requisitos de usabilidad y sencillez de manejo de la aplicación combinado con un interfaz ameno. Casi todos los usuarios que probaron CLOC no tuvieron problemas en su manejo, exceptuando el usuario 3. La razón se puede encontrar en su avanzada edad, 82 años, y el no cumplimiento de tener algún conocimiento mínimo de informática, puesto que no había usado un ordenador nunca antes. De hecho, fue el único usuario que pidió asistencia durante la realización de las tareas.

Todos los usuarios coincidían en que CLOC es una aplicación muy útil y de sencillo manejo, y 8 de los 10 (80%) la usarían en casa para gestionar sus libros, CDs y películas.

Es importante destacar que todos los usuarios coincidían en que una vez utilizada la aplicación por primera vez, el resto de veces que la utilizaban era todavía más sencillo y amigable, y podrían realizar las tareas incluso en menos tiempo.

Respecto a trabajo futuro, en primer lugar, se pretende aplicar los comentarios proporcionados por los usuarios durante la evaluación de usabilidad de CLOC. Así, por ejemplo, la mayoría de los usuarios coincidían en que a la hora de realizar acciones como eliminar, modificar o ver la carátula de un libro, CD o película, tener que seleccionar el atributo **id_objeto** era una falta de usabilidad. De hecho, fue gracias a los mensajes de ayuda que pudieron manejar correctamente la aplicación.

Otras posibilidades interesantes que se podrían intentar con CLOC es permitir a los usuarios personalizar la interfaz, eligiendo otra imagen de fondo, e incluyendo opciones de accesiblidad para aumentar el tamaño y tipo de las fuentes. También se podría ampliar la capacidad de gestión de CLOC a otro tipo de objetos, como zapatos o ropa.

6. Bibliografía

(Cadenhead & Lemay, 2008) Rogers Cadenhead & Laura Lemay, 2008- Programación Java 6, Edición Anaya Multimedia.

(Ceballos, 2007) Francisco Javier Ceballos, 2007- Programación orientada a objetos con C++ - 4 edición, Edición RA-MA

(Loney, 1995) Kevin Loney, 1995- Oracle. Manual del adminsitrador, Edición McGraw-Hill.

(Louden, 2004) Kenneth C. Louden, 2004- Lenguajes de Programación – principios y práctica, Edición Thomson.

(Nielsen, 2006) Nielsen, 2006- Usabilidad, Edición Anaya Multimedia.

(Pérez, 2002) Cesar Pérez, 2002- Oracle 9i. Administración y Análisis de Bases de Datos, Edición RA-MA.

(Pérez, 2007) Cesar Pérez, 2007- MySQL para Windows y Linux 2ª Edición, Edición RA-MA.

(Pressman, 2002) Roger S. Pressman, 2002- Ingenieria del Software, Edición Mc GrawHill.

(Summerfield, 2009) Mark Summerfield, 2009- Programación Python 3, Edición Anaya Multimedia .

(Training Solutions, 2004) Online Training Solutions Inc, 2004 –Paso a Paso. Microsoft Office Access 2003, Edición McGraw-Hill.

(Tucker & Noonan, 2003) A. Tucker & R. Noonan, 2003-Lenguajes de programación-principios y paradigmas, Edición Mc Graw Hill.

Anexo 1. Descripción exhaustiva de la Base de Datos

Ahora se procederá al detalle de cada uno de los elementos que la componen:

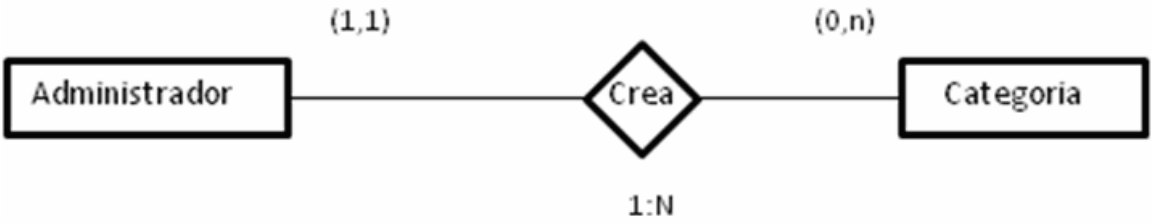

Figura 1. Gráfico de la relación CREA

ADMINISTRADOR (Id_Admin, NIF, NombreA, Apellido 1, Apellido2, Correo, Password)

El administrador se encargada de crear las categorías que van a componer la base de datos. Con esta relación (CREA) se puede saber que administrador/es han creado la/s categoría/s. Aunque nuestro administrador se crea por defecto ya que la aplicación es local, al principio se pensó en que hubiese más de una administrador y así cada administrador poder gestionarlas categorías, pero eso era para la aplicación Web Ahora detallamos los atributos de la tabla ADMINISTRADOR:

6. **Id_Admin:** Actúa como un campo numérico secuencial, que hace de clave primaria en la tabla.
7. **NIF**: Actúa como alternativa. Por lo tanto es único para cada administrador. Está compuesto por nueve números y una letra.
8. **NombreA**: Almacena el nombre de dicho administrador, sin espacios
9. **Apellido 1**, **Apellido2**: Almacena sus apellidos por separado. Cada apellido en un atributo, sin espacios.
10. **Correo**: En este campo se almacena una dirección de correo electrónico, que sirve para mandar información a dicho administrador.
11. **Password**: Clave de acceso que es utilizada por el administrador para entrar en el sistema.

La categoría se utiliza para referenciar los distintos catálogos de los que va a disponer el sistema. Para extender esta información, y dar unas claves de su funcionamiento, se dirá que la categoría número uno, se puede hacer corresponder con las películas, la categoría numero dos con los CDS de música, y así sucesivamente con cuantas categorías se almacenen en el sistema.

CATEGORIA (ID-CAT, NombreC, Descripcion, Admin↑)

1. **ID-CAT**: Almacena un identificador único para cada categoría. Será un numero secuencial, para así tenerlo siempre identificado.
2. **NombreC**: Almacena el nombre que corresponde con dicha categoría.

3. **Descripcion**: Este campo almacena algún comentario (opcional) del administrador que creó la categoría. Será un comentario de dicho administrador a la hora de crear dicha categoría. Puede contener espacios o no contener nada, para dar esa propiedad de opcional.
4. **Admin**: Clave Ajena a la tabla Administrador. Con ella, obtenemos el Administrador que ha creado la categoría. Esta valor contendrá el **contador** (clave primaria de la tabla ADMINISTRADOR) que será obligatorio que este valor esté en la tabla ADMINISTRADOR (Regla de integridad referencial).

Las cardinalidades de la relación CREA, se explican de la siguiente manera:

- **(1,1)** – Una categoría se crea una única vez y por un único administrador. No se puede hacer dos categorías iguales.
- **(0,n)** – Ningún administrador está forzado a que tenga que crear alguna categoría. Pero un administrador puede crear ninguna o muchas categorías.

En la relación CREA se almacena la categoría y por quien fue creada. Se continúa con la descripción de la relación TIENE, como se muestra en la Figura 35.

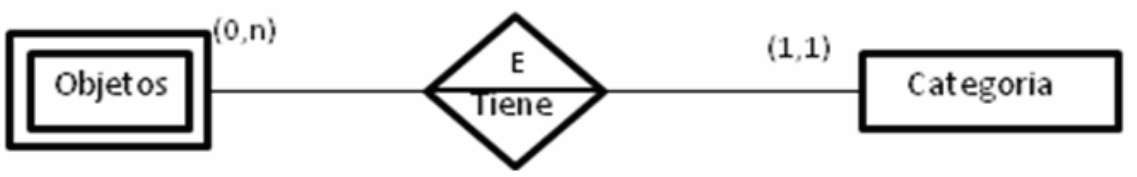

Figura 2. Gráfico de la relación TIENE

OBJETO (<u>ID-OBJ</u>, ID-CAT2↑)

Esta relación (en extensión) almacena cada objeto a una categoría, por lo que se tienen todos los objetos que componen cada categoría. Esto, informa que un libro y una película pueden llamarse igual, pero uno de ellos pertenece a la categoría libro, y el otro a la categoría película, por lo que son objetos distintos. Los atributos del súper-tipo objeto son los siguientes.

1. **ID-OBJ**: Atributo que almacena un numero a un objeto, creándose dicho numero de manera secuencial por objeto existente. De esta manera, no puede existir dos objetos con el mismo identificador, y así actúa de clave primaria.
2. **ID-CAT2**: Este atributo, se necesita por ser OBJETO una entidad débil y por reflejar la relación en extensión. Por lo tanto, este atributo es el mismo que se tenga en la CATEGORIA en su campo ID-CAT. Por lo tanto se está actuando de clave ajena. A este campo no se le va a permitir el valor nulo, ya que se tiene siempre que un objeto pertenece a una categoría. Se refleja también que si se borra una categoría, se borren todos los objetos de dicha categoría.

Las cardinalidades de la relación TIENE son las siguientes:

- **(0,n)** – Una categoría permitimos que tenga desde cero objetos hasta n. Permitimos el cero, debido a que una categoría inicialmente no tiene ningún objeto.

- **(1,1)** – Un objeto determinado, identificado por su clave primaria, solo puede pertenecer a una categoría y solo a una.

COMPONENTES DE OBJETO

Posteriormente, se tiene una generalización, que a la vez esta forzada a que sea exclusiva, por la dependencia comentada anteriormente. Un objeto, solo puede pertenecer a una categoría, y pertenece a una de ellas. Esto es así, debido a que un objeto en particular está asociado a una categoría. Nos puede llevarse a confusión el acto de que una película y un libro (por ejemplo) se llamen igual. Pero son objetos diferentes, y cada uno de ellos pertenece a un subtipo y solo a uno. El libro pertenece a los libros y la película a las películas. Con esta doble restricción para los objetos (una por parte de la relación tiene, y otra por la generalización del objeto) conseguimos mejorar el rendimiento del sistema.

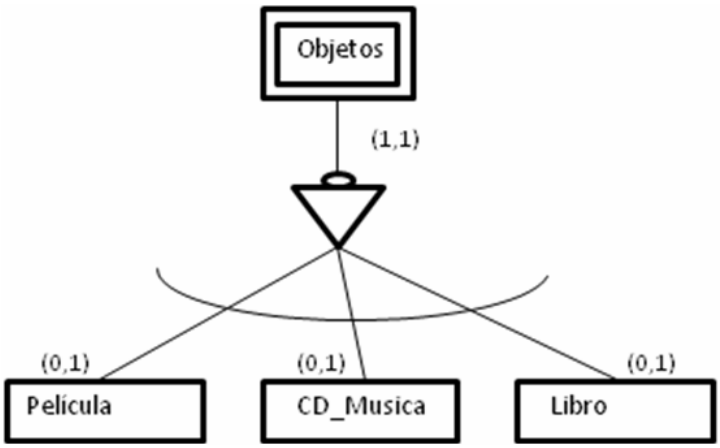

Figura 3. Gráfico de la generalización de OBJETO

Cada sub-tipo tiene unos atributos únicos. No se almacena lo mismo en una película que en un libro, pero si tienen todos en común que se almacena la información del súper-tipo OBJETO.

A continuación se detalla la información de cada uno de ellos.

CD_MUSICA (ID-OBJ↑, Título, ID-CAT↑,Autor, Num-Discos, Caratula_F, Caratula_B, Género)

1. **Titulo**: Almacena el título del disco de música. Pertenece a la clave primaria, junto con el ID_OBJ.
2. **Autor**: Corresponde con el autor de dicho disco.
3. **Num-Discos**: Numero de discos (CDS) que contiene el titulo.
4. **Caratula_F**: Almacena la imagen redimensionada de la carátula frontal del título.
5. **Caratula_B**: Almacena la imagen redimensionada de la carátula trasera del título.
6. **Género**: Información relativa al género del título.

PELICULA (ID-OBJ↑, ID-CAT↑,Nombre, Num-s, Caratula, Formato, PDirector)

1. **Nombre**: Nombre de la película

2. **Num-s**: Almacena el número de elementos de soporte para almacenar dicha película
3. **Caratula**: Almacena la imagen redimensionada para poder visualizar la carátula de dicha película.
4. **Formato**: Actualmente, para las películas existen diversos formatos, tales como DVD, Blu-Ray o en su caso VHS. Aquí se recogen todos los formatos en los que se puede almacenar dicha película.
5. **PDirector**: Almacena el nombre del director de la película.

LIBRO (ID-OBJ↑, ID-CAT↑, Titulo, LAutor, Sinopsis, ISBN, Num_Edicion, Caratula)

1. **Titulo**: Contiene el título del libro.
2. **LAutor**: Contiene el nombre del autor.
3. **Sinopsis**: Breve resumen del libro
4. **ISBN**: Número de identificación único que existe en los libros.
5. **Num_Edicion**: Contiene el numero de la edición del libro.
6. **Caratula**: Almacena la ruta de la imagen correspondiente a la carátula del libro.

Los subtipos, se definen debido a que existen grandes diferencias entre ellos. Los atributos (información a almacenar) son distintos, y uno de ellos en este caso, los CD de música tiene una relación en identificación, por lo que esta generalización aclara mucho el esquema.

Otra característica importante, es que a la hora de que se cree otra categoría, con el esquema que se ha propuesto, solo bastaría con crear la correspondiente tabla en objeto. La generalización crecería abarcando más subtipos, con lo que la modificación de la base de datos sería mínima, con lo que se adaptaría mejor a futuros cambios.

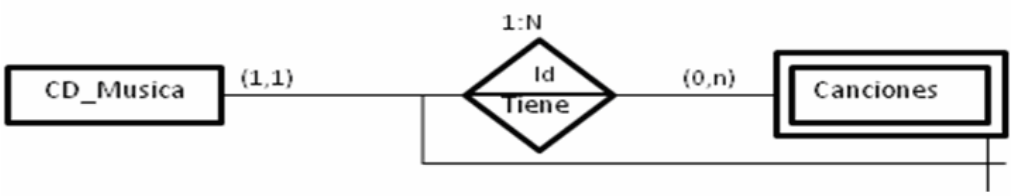

Figura 4. Gráfico de la relación TIENE entre las tablas CD_MUSICA y CANCIONES.

Aunque esta tabla no se haya utilizado en CLOC, se mantiene por posibles extensiones futuras. La última característica de los CD_MUSICA es que tienen canciones. Con lo que es necesario almacenar las canciones que tienen cada uno de los elementos de la tabla CD_MUSICA. Como se observa en la imagen, existe una dependencia en identificación para identificar una canción. Una canción se identifica por medio del campo ID-OBJ que será único por cada ocurrencia del la tabla CD_MUSICA y por otros campos de la misma tabla. Después tendremos los campos propios de una canción que se detallan a continuación:

CANCIONES (ID-OBJ↑, EnCD, Track, Titulo, NCancion, Interprete)

1. **ID-OBJ**: atributo que se propaga de la tabla CD_MUSICA.
2. **Título:** atributo que también se propaga de la tabla CD_MUSICA.
3. **NCanción:** atributo en el que se introduce el nombre de la canción
4. **Interprete:** Autor de la canción. Este campo puede tener para una misma ocurrencia de un cd de música elementos distintos (recopilatorio) o ser siempre el mismo (cuando es de un autor).

5. **EnCD:** Indica en el disco que esta la canción.
6. **Track:** Nos informa del número de canción dentro del cd.

La clave primaria, para identificar una canción, se pensaría que podría estar formada por los atributos **ID-OBJ** y **Título**. Para ubicar una canción, se necesita los campos **EnCD** y **Track**, que dicen exactamente donde está ubicada dicha canción. Sabiendo esto, se ve que si identifica una canción por el ID-OBJ y el titulo, solo se podría introducir una única canción. En cambio, si se introduce por el **ID-OBJ** y los campos numéricos de ubicación del la canción, se tendrá siempre identificada, y se asegura a su vez que no se introduce una canción dos veces, ya que la clave primaria no puede estar repetida.

Las cardinalidades de la relación TIENE se definen de esta manera:

- **(0,n)** – Indica que un CD puede tener de cero canciones a muchas. El cero se permite debido a que al introducir CD de música por primera vez, este no tendrá canciones y se tendrán que agregar después si es que se agregan. Posteriormente un elemento podrá tener muchas canciones.
- **(1,1)** – Una canción pertenece únicamente a una ocurrencia de CD_MUSICA. Esto es así, debido a que no existen dentro de la tabla CD_MUSICA ocurrencias que tengan el mismo ID-OBJ. Por lo tanto cada canción irá a una ocurrencia. Cuando se tenga elementos en canciones que tengan el mismo nombre de canción el mismo interprete, será la única información que sea igual. La clave primaria será diferente, y los atributos destinados para localizar la canción también serán distintos.

USUARIO (<u>Id_usuario</u> , Alias, Nombre apellido1, apellido2, correo, password)

El usuario es sin duda la parte más importante, ya que son los que realmente van a poder usar la aplicación. Está involucrado en más de un relación como se muestra en la imagen siguiente.

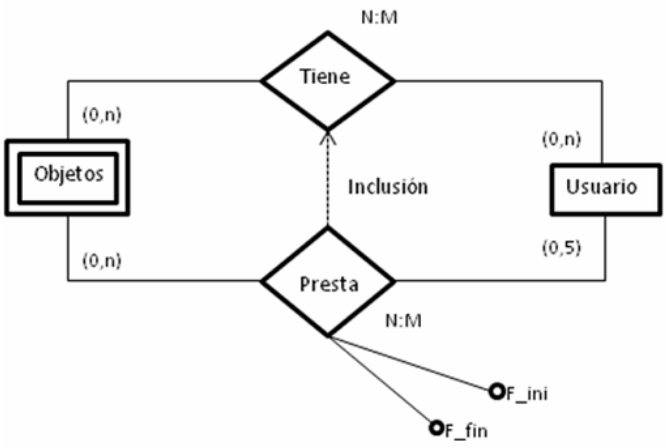

Figura 5. Gráfico en el cual aparece involucrado la tabla USUARIO.

Los atributos de usuario son los siguientes:

1. **Id_Usuario:** campo numérico secuencial, para identificar (clave primaria) a un usuario de manera única.
2. **Alias:** Atributo alfanumérico por el cual se identifica al usuario de manera única. Este atributo va a ser la clave primaria.
3. **Nombre:** El nombre real del usuario. Contendrá una cadena de caracteres sin espacios.
4. **Apellido1, Apellido2:** Cadena de caracteres que contendrá los apellidos del usuario.
5. **Correo:** Contendrá el correo electrónico que facilite el usuario. La tarea de comprobar el formato y validez del correo será externo a la base de datos. Se recuerda que el formato de un correo electrónico viene dado por la siguiente regla: caracteres@caracteres.caracteres. La longitud de los caracteres también será comprobada.
6. **Password:** Contendrá una cadena alfanumérica para que el usuario pueda entrar en el sistema. Inicialmente al usuario se le proporciona una clave automática. Dicha clave será generada externamente en la base de datos, pero almacenada en ella.

Una vez descritos los atributos de la tabla USUARIO, según la imagen de la figura 38, se relaciona con la tabla objeto. Es lógico pensar esto, debido a que realmente el usuario almacena los objetos, lo que se contempla en la relación TIENE, y que dicho usuario presta sus objetos contemplado en la relación PRESTA. Como vemos, las relaciones en sí están también relacionadas por inclusión. Esto nos informa únicamente que un usuario no puede prestar un objeto que no tenga en su catalogo. Lo que nos muestra una restricción. De manera resumida, un usuario no puede prestar algo que no tiene.

Recordamos a continuación los atributos de la tabla OBJETO que son los siguientes.

OBJETO (ID-CAT2↑, ID-OBJ)

Ahora se detalla la relación tiene (Figura 38). Como se ha descrito, el usuario tiene objetos, y puede tener esos objetos de distintas categorías. No se limita a que un usuario puede tener solo objetos de una categoría, puede tenerse de todas las que existan, o más especifico, todas las categorías que los administradores quieran. Así que un usuario tiene objetos, y los objetos los tienen los usuarios. Esta frase sirve para detallar las cardinalidades de la relación, para así terminar las restricciones de la relación.

- **(0,n)** – Un usuario tiene como mínimo cero objetos. Esta situación se da al usuario darse de alta, no se le introduce ningún elemento de prueba o de muestra. Como mucho tendrá **n** objetos, que será el tamaño total de objetos que tenga en su colección o catalogo.
- **(0,n)** – Un objeto lo puede tener ningún usuario. Hay que recordar que primero se dan de alta los objetos, y después son los usuarios quienes los recogen en el conjunto de su catalogo. Posteriormente, ese mismo objeto lo pueden tener catalogado muchos usuarios, es por eso la cardinalidad máxima **n**.

Como se ve por las cardinalidades y por el gráfico de la Figura 21, estamos ante una relación N:M. Este tipo de relaciones tienen un tratamiento especial a la hora de crear las tablas y su paso a MySQL.

Para representar este tipo de relaciones hay que apoyarse en una tabla auxiliar, que contendrá una clave ajena por cada una de las tablas, y la unión de ambas formarán la clave primaria.

USUARIO_OBJETO(ID_OBJ↑,Id_Usuario↑)

Como se ve en este ejemplo, la tabla USUARIO_OBJETO almacena únicamente las claves primarias de las tablas a las que hace referencia, y así hace la función de tabla auxiliar. Es por eso, que ambos campos, son claves ajenas:

1. **ID_OBJ:** Es una clave ajena, que con ella se puede obtener todos los datos del objeto.
2. **Id_Usuario:** También es clave ajena, pero esta vez nos identifica a cada usuario. Con lo que con ella se puede obtener todo lo referente a un usuario.
3. La unión de ambos atributos, será la clave primaria, para así saber todos los objetos que tiene un usuario, la descripción del objeto, el catalogo al que pertenece, y los propios datos del usuario.

Se observa que mediante esta tabla **usuario_objeto**, se tienen los datos que se pretenden tener mediante la relación tiene de la Figura 38. Se sabe, a su vez que se puede representar también todos los objetos que tiene un usuario, y también todos los usuarios que tienen un objeto. Este es el objetivo que se tenía que conseguir.

PRESTAMO_USUARIO
(Id_Prestamo,id_prestamista↑,Id_Objeto↑,NombreP,Fecha_Inicio,Fecha_FIN)

1. **Id_Prestamo:** Será un numero secuencial que nos identificará cada préstamo, por lo que ejerce de clave primaria en la tabla.
2. **Id_prestamista:** Esta campo es una clave ajena a la tabla usuario. Informa que usuario del sistema realiza un préstamo.
3. **Id_Objeto:** Este campo es también una clave ajena, pero a la tabla objeto. Con lo que se obtiene toda la información del objeto que el prestamista está prestando.
4. **NombreP:** Indica mediante una cadena de caracteres a la persona a la que se presta el objeto en cuestión. Aquí no se hace referencia a que sea un usuario que exista. Esto es debido a que el usuario puede prestar un objeto a una persona que no esté dada de alta en nuestro sistema.
5. **Fecha_Inicio:** Campo de tipo fecha, para almacenar la fecha de inicio del préstamo.
6. **Fecha_Fin:** Campo también de tipo fecha, para almacenar el fin del préstamo.

Por su parte, la relación presta es más restrictiva, ya que solo se pueden prestar objetos que el usuario tenga en la relación tiene. Esta parte se puede controlar por la base de datos (se sobrecargaría de operaciones) o por medio de JAVA.

La cardinalidades de dicha relación se detalla a continuación.

- **(0,n)** – Esta parte nos informa, de que un objeto lo puede prestar cualquier usuario, si es que lo presta. El usuario puede no prestarlo (se refiere al conjunto de todos los usuarios) o puede prestarlo más de un usuario.
- **(0,5)** – Esta restricción en la cardinalidad está fijada en los requisitos del sistema. Un usuario puede no prestar ningún objeto, o prestar como máximo cinco objetos de cualquier categoría.

Como se ve, para recoger los préstamos, se necesita una fecha de inicio del préstamo, y otra fecha de finalización del mismo. Es por esto que dichos atributos pertenecen a la relación. Esta información se recoge el día que el usuario lo presta, y que el usuario vuelve a tener dicho objeto en su poder. Estos atributos son **Fecha_Inicio** y **Fecha_Fin**. De manera aclaratoria también se introduce un atributo **Nombre_P** que indica a quien se ha prestado dicho objeto.

Nivel Físico

A continuación se define el nivel físico de las tablas, con su correspondiente código.

La tabla administrador se crea con el código descrito a continuación. Se ve que todos sus campos no admiten que no tengan valor. A su vez, se tienen dos campos especiales, que además de no permitir valores nulos, estos deben de ser únicos. Estos campos son el NIF y el correo, que se sabe por el mundo real que tienen que ser únicos por persona.

Se observa también que la clave primaria contiene un número que va creciendo de forma automática. Esto, se observa en muchas tablas.

```
CREATE TABLE Administrador
(
    Id_Admin INT AUTO_INCREMENT PRIMARY KEY,
    NIF VARCHAR(9) NOT NULL,
    Nombre VARCHAR(20) NOT NULL,
    Apellido1 VARCHAR(20) NOT NULL,
    Apellido2 VARCHAR(20) NOT NULL,
    Correo VARCHAR(50) NOT NULL,
    Password VARCHAR (15) NOT NULL,
    UNIQUE(Correo),
    UNIQUE (NIF)
);
```

La siguiente tabla será la categoría. Dicha tabla tiene una definición de clave ajena. Para ello se utiliza en la tabla el ENGINE InnoDB para efectuar la regla de integridad referencial. Como se ve, la descripción de la categoría puede contener un valor nulo. Los demás campos, no pueden ser nulos. El campo Admin, tampoco, ya que tiene que contener un valor tal, que este en la tabla administrador.

```
CREATE TABLE Categoria
(
    ID_CAT INT AUTO_INCREMENT PRIMARY KEY,
```

```
    NombreC VARCHAR(30) NOT NULL,
    Descripcion VARCHAR(300),
    Admin int NOT NULL,
    KEY (Admin),
    FOREIGN KEY (admin) REFERENCES Administrador (Id_Admin)
ON DELETE RESTRICT ON UPDATE RESTRICT)
ENGINE=InnoDB;
```

La tabla Objeto, es sin lugar a dudas una de las más importantes. Se ha creado una tabla solo para el súper-tipo debido a análisis del mundo real. Si se implementaran más subtipos, lo que habría que definir mas categorías, habría que reprogramar ciertos criterios. De esta manera, al tener una tabla genérica, se pueden crear más sub-tipos independiente de los ya creados.

Se ve que solamente tiene dos campos. Un campo auto numérico por cada objeto creado y una clave ajena para obtener la categoría a la que pertenece el objeto, para así saber a qué tabla pertenece.

```
CREATE TABLE Objeto
(
    ID_CAT2 INT NOT NULL,
    ID_OBJ INT AUTO_INCREMENT PRIMARY KEY,
    KEY (ID_CAT2),
    FOREIGN KEY (ID_CAT2) REFERENCES Categoria (ID_CAT)
ON DELETE CASCADE ON UPDATE CASCADE)
ENGINE=InnoDB ;
```

La tabla CD_Musica , depende del súper-tipo objeto OBJETO , es por esto que tiene dos campos que son claves ajenas, uno por la tabla objeto (por su ID_OBJ) y otro por obtener el catalogo al que pertenece. Se ve como elemento destacable que la clave primaria esta formada por dos campos, el propio ID_OBJ, y el titulo del CD, para así no permitir dos tuplas iguales en su nombre. Es imposible por el mundo real, que dos discos se llamen igual.

```
CREATE TABLE CD_MUSICA
(
    ID_OBJ int NOT NULL,
    ID_CAT int NOT NULL,
    Titulo VARCHAR(50) NOT NULL,
    Autor VARCHAR(50) NOT NULL,
    Num_Discos int NOT NULL,
    Caratula_f MEDIUMBLOB,
    Caratula_b MEDIUMBLOB,
    Genero VARCHAR(20) NOT NULL,
    Primary key (ID_OBJ,Titulo),
    key (ID_CAT),
    FOREIGN KEY (ID_CAT) REFERENCES Categoria(ID_CAT)
    ON DELETE CASCADE ON UPDATE CASCADE,
    key (ID_OBJ),
    FOREIGN KEY (ID_OBJ) REFERENCES Objeto(ID_OBJ)
ON DELETE CASCADE ON UPDATE CASCADE)
```

```
ENGINE=InnoDB;
```

La película también tiene definidas las mismas claves ajena que los CDS de música.

```
CREATE TABLE PELICULA
(
    ID_OBJ int PRIMARY KEY,
    ID_CAT int NOT NULL,
    Nombre VARCHAR(150) NOT NULL,
    Num_S int NOT NULL,
    Caratula MEDIUMBLOB,
    Formato VARCHAR(20) NOT NULL,
    PDirector VARCHAR(80),
    key (ID_CAT),
    FOREIGN KEY (ID_CAT) REFERENCES Categoria(ID_CAT)
    ON DELETE CASCADE ON UPDATE CASCADE,
    key (ID_OBJ),
    FOREIGN KEY (ID_OBJ) REFERENCES Objeto(ID_OBJ)
ON DELETE CASCADE ON UPDATE CASCADE)
ENGINE=InnoDB;
```

La tabla libro, tiene la misma estructura que la película, en cuanto a la clave primaria, y las claves ajenas. Lógicamente, un libro tiene sus campos propios.

```
CREATE TABLE LIBRO
(
    ID_OBJ int PRIMARY KEY,
    ID_CAT int NOT NULL,
    Titulo VARCHAR(150) NOT NULL,
    Lautor VARCHAR(150) NOT NULL,
    Sinopsis VARCHAR(700),
    ISBN VARCHAR(25),
    Num_Edicion int ,
    Caratula MEDIUMBLOB,
    key (ID_CAT),
    FOREIGN KEY (ID_CAT) REFERENCES Categoria(ID_CAT)
    ON DELETE CASCADE ON UPDATE CASCADE,
    key (ID_OBJ),
    FOREIGN KEY (ID_OBJ) REFERENCES Objeto(ID_OBJ)
ON DELETE CASCADE ON UPDATE CASCADE)
ENGINE=InnoDB;
```

Una canción depende de una tupla de música. Por lo tanto, siempre se agregarán canciones a un titulo determinado. Es por eso que el título pertenece a la clave primaria del elemento música. Respecto a las canciones, en el mundo real, distintos títulos pueden tener alguna canción que se llame igual. Por eso, una canción se identifica con el objeto, y la posición que ocupa. De esta manera tampoco puede existir en el mismo titulo dos canciones en la misma posición. Si un disco de música, por ejemplo, tiene 12 canciones, no podrá tener dos canciones que sean la primera, ya que daría un error en la clave primaria.

```
CREATE TABLE CANCIONES
(
    ID_OBJ int,
    EnCD int,
    Track int,
    Titulo VARCHAR(50) NOT NULL,
    Ncancion VARCHAR(150),
    Interprete VARCHAR(150),
    PRIMARY KEY(ID_OBJ,EnCD,Track),
    key (ID_OBJ,TITULO),
    FOREIGN KEY (ID_OBJ,TITULO) REFERENCES
    CD_MUSICA(ID_OBJ,TITULO)
ON DELETE CASCADE ON UPDATE CASCADE)
ENGINE=InnoDB;
```

El usuario, se ve que tiene una definición muy parecida a la del Administrador. Se ve que el Alias no admite valores nulos y además estos han de ser únicos. Muestra que es una clave alternativa. Como se puede ver el usuario es una tabla que no tiene claves ajenas.

```
CREATE TABLE Usuario
(
    Id_Usuario INT AUTO_INCREMENT PRIMARY KEY,
    Alias VARCHAR(30) UNIQUE NOT NULL,
    Nombre VARCHAR(20) NOT NULL,
    Apellido1 VARCHAR(20) NOT NULL,
    Apellido2 VARCHAR(20)NOT NULL,
    Correo VARCHAR(50) NOT NULL UNIQUE,
    Password VARCHAR(10) NOT NULL
);
```

Esta tabla se crea al estar en una relación N:M. Como se comentó en el esquema, ejerce de tabla auxiliar. En ella se almacenan dos campos que serán clave primaria, y cada uno de ellos será una clave ajena a cada tabla definida en FOREIGN KEY.

```
CREATE TABLE Usuario_objeto
(
    ID_OBJ int NOT NULL,
    id_Usuario int NOT NULL,
    primary key(ID_OBJ,Id_Usuario),
    key (ID_OBJ,Id_Usuario),
    FOREIGN KEY (ID_OBJ) REFERENCES Objeto(ID_OBJ)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (Id_Usuario) REFERENCES Usuario(Id_Usuario)
ON UPDATE CASCADE ON DELETE CASCADE)
```

```
ENGINE=InnoDB;
```

El préstamo, al querer almacenar un histórico de los préstamos de cada usuario, la forma más eficiente de identificarlos será con un único campo. De esta manera, un usuario puede prestar un mismo objeto, a la misma persona, pero solo será diferente las fechas. Se ve que la fecha de fin del préstamo (Fecha_Fin) es el único campo que admite valores nulos. Esto es lógico pensarlo ya que en el mundo real prestas un objeto en una fecha de inicio, pero no sabes cuando ese usuario o persona te va a devolver dicho objeto. Así cuando finalice el préstamo, se insertará la fecha de devolución (Fecha_Fin).

```
CREATE TABLE PRESTAMO_USUARIO
(
    Id_Prestamo int AUTO_INCREMENT PRIMARY KEY,
    id_Prestamista int NOT NULL,
    ID_OBJ int NOT NULL,
    NombreP VARCHAR(100) NOT NULL,
    Fecha_Inicio DATE NOT NULL,
    Fecha_Fin DATE,
    key(Id_Prestamista,ID_OBJ),
    FOREIGN KEY (Id_Prestamista) REFERENCES USUARIO(Id_Usuario)
    ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (ID_OBJ) REFERENCES Objeto(ID_OBJ)
ON UPDATE CASCADE ON DELETE CASCADE)
ENGINE=InnoDB;
```

Anexo 2. Hoja de tareas

1. Insertar un Libro
2. Modificar ese libro
3. Buscar ese libro
4. Consultar Películas Disponibles
5. Consultar Libros Disponibles
6. Consultar CDS Disponibles
7. Eliminar el Libro introducido

Tabla de Objetivos (Realizadas/no Realizadas)

	Tarea 1	**Tarea 2**	**Tarea 3**	**Tarea 4**	**Tarea 5**	**Tarea 6**	**Tarea 7**	**M**
U1								
U2								
U3								
U4								
U5								
U6								
U7								
U8								
U9								
U10								
M								

Anexo 3. Cuestionario de satisfacción

Indique la valoración de 1 a 10 en las preguntas que el administrador le indique.

1. ¿Cuál es su nombre?
2. ¿Qué edad tiene?
3. ¿Le ha gustado la aplicación?
4. ¿Qué estudios tiene?
5. ¿Cuántas horas usa al día el ordenador?
6. ¿Qué le ha parecido la aplicación?
7. ¿Cree que la aplicación es útil?
8. ¿Recomendaría la aplicación a algún conocido suyo?
9. En general, ¿le gusta usar el ordenador o prefiere realizar las tareas a mano?
10. ¿Que le parece mejor la imagen del árbol o una imagen con direcciones de bienvenida?
11. ¿Prefiere introducir los datos en la etiqueta antes del botón o después del botón?
12. ¿Le ha resultado difícil manejar la aplicación?
13. ¿Ha usado la ayuda? En caso afirmativo, ¿Qué le ha parecido? En caso contrario, ¿Por qué no la ha usado?
14. ¿Qué cambiaria de la aplicación?
15. ¿Ha tenido problemas en realizar algún de las tareas?
16. ¿Cuál ha sido la tarea más sencilla? ¿Que cree que se lo ha facilitado? Consulta.
17. ¿Cuál ha sido la tarea más compleja? ¿Qué dificultades ha encontrado?
18. ¿Cree que es necesario un sistema de este tipo?
19. ¿Cree que lo usarían en casa?