Universidad
Carlos III de Madrid
www.uc3m.es

Universidad
Carlos III de Madrid

TRABAJO FIN DE GRADO

DESARROLLO DE SISTEMAS DE INFORMACIÓN CON FRAMEWORK SPRING-HIBERNATE.

PRUEBA DE CONCEPTO: DESARROLLO DE UN SISTEMA DE PRÉSTAMO BIBLIOTECARIO

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Raúl Rivas Greciano

Tutor: Miguel Ángel Patricio

Colmenarejo, Septiembre de 2013

AGRADECIMIENTOS:

Mis más sinceros agradecimientos a:

A mi familia, en especial a mis padres, por haberme brindado la oportunidad de estudiar una carrera y apoyarme en todo momento para seguir adelante.

A mis compañeros de clase, por haberme mostrado su apoyo en momentos difíciles y por ayudarme a esforzarme en ser en un futuro un mejor Ingeniero Informático.

A mi tutor Miguel Ángel Patricio, por haber estado tan atento a mi trabajo y por haberme dado todo su apoyo y depositado su confianza en mí.

A mis profesores, por todos los conocimientos que me han proporcionado a lo largo de estos años, tanto en el aspecto académico como en el personal.

A mis amigos, por aguantar mis encierros de fin de semana para terminar el proyecto y por mostrar interés cuando les hablaba de mi proyecto.

Mi novia, por aguantar mi estrés en ciertos momentos y estar ahí siempre que la necesitaba.

RESUMEN:

En el presente Trabajo de Fin de Grado que se aborda se pretende acercar el uso del Framework Spring y el ORM Hibernate a cualquier persona que se quiera iniciar en cualquiera de las dos tecnologías Java.

Actualmente ambas tecnologías tienen una gran aceptación en los entornos de desarrollo de muchas empresas y su uso se ha incrementado enormemente en los últimos años gracias a su política de código abierto.

A lo largo de este Trabajo de Fin de Grado estudiaremos ambas tecnologías; analizando los puntos fuertes y débiles tanto de Spring como de Hibernate y una vez realizado un estudio a fondo de ambas tecnologías, se crearán sendos tutoriales estudiando su funcionamiento por separado proporcionando un primer contacto con estas tecnologías. Finalmente se creará una prueba de concepto más completa y compleja en el que mostraremos el uso de ambas tecnologías en conjunto.

Índice:

Índice de imágenes:

Índice de tablas:

1. INTRODUCCION

En este capítulo inicial se expone la visión general del Trabajo de Fin de Grado, cuáles han sido las principales motivaciones para realizarlo, por qué puede resultar útil y qué objetivos pretenden alcanzarse.

1.1. Motivación:

En el presente Trabajo se pretende realizar un estudio a fondo de las tecnologías Hibernate y Spring.

La finalidad del desarrollo de ambas tecnologías que se realizará es dar un tutorial paso a paso del funcionamiento y uso de las mismas. De esta forma se conseguirá facilitar la introducción a ambas tecnologías y de igual modo se promueve una mayor expansión de su uso.

Para mejor comprensión, es muy recomendable poseer un conocimiento medio o alto del lenguaje de programación Java por parte del lector. Esto es debido a que habrá conceptos propios de Java tales como objetos, métodos o uso de variables entre otros que se darán por conocidos para poder avanzar en el uso de las tecnologías citadas anteriormente.

Para finalizar el estudio de Hibernate y Spring se realizará una prueba de concepto sobre un préstamo bibliotecario en el que se mostrarán muchos de los puntos fuertes de ambas tecnologías tales como: Inyección de dependencias de clase, conversión de tablas de base de datos en listas de objetos Java, definición de transacciones…

A continuación se procede a entrar en detalle de porque se decidió usar cada una de las tecnologías presentes en el proyecto.

1.1.1. Framework, Spring

La gran mayoría de los Framework para el lenguaje de programación Java tienen como función principal simplificar el desarrollo de aplicaciones Java. Pero en la realidad muy pocos Framework consiguen realmente hacer la programación más sencilla debido a normalmente los otros Frameworks piden que las clases Java de tu programa hereden clases propias del Framework o implementen unas interfaces específicas para su funcionamiento. El principal problema de todo esto es que si en algún momento se decide dejar de utilizar dicho Framework se tienen que modificar todas las clases Java del proyecto, con un enorme coste de mantenimiento en la aplicación.

Para prevenir este problema surgió el Framework Spring. En un artículo Sobre Spring[1] lo definen de la siguiente forma: "Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications"; "El Framework Spring es una plataforma Java que actúa de soporte y proporciona una infraestructura para las aplicaciones Java", es decir, es el encargado de realizar las interacciones entre los diferentes componentes que posee la aplicación.

La principal potencia de Spring es que las clases que lo usan no necesitan ningún tipo de programación extra, por lo que puedes centrarte en crear tu aplicación y Spring se encargara del resto. Otra de las grandes ventajas de Spring es poder realizar Inyección de dependencias, con lo que se consigue eliminar las dependencias entre clases ya que no se llamarán entre ellas, sino que Spring el hará el canal de comunicación.

[1] http://static.springsource.org/spring/docs/3.1.0.M2/spring-framework-reference/html/overview.html

1.1.2. Interacción con Base de Datos, Hibernate

Para conectar el programa con la base de datos se decidió usar una herramienta del tipo ORM (Object-Relational mapping) en vez del JDBC convencional. La principal característica de un ORM consiste en poder usar las tablas de la Base de Datos como objetos en Java, entre los más conocido y usados se encuentran Hibernate y JPA.

Se decidió usar el ORM Hibernate ya que es una tecnología muy usada en entornos laborales y mediante anotaciones se puede realizar fácilmente el mapeo de atributos de una Base de Datos relacional al modelo de objetos de la aplicación Java.

Hibernate usa una gestión de sesiones que posee métodos propios con los cuales hacer acciones simples como puede ser el agregado, la modificación o un select a la Base de Datos. Por otro lado, Hibernate también ofrece un lenguaje propio llamado HQL (Hibernate Query Language) con el que poder realizar consultar personalizadas.

1.1.3. Interfaz gráfica, Swing

En la actualidad la gran mayoría de aplicaciones se crear para poder ejecutarlas desde un navegador. De esta forma puede llegar a un gran número de personas sin tener que difundir el programa en sí. Esta es la principal razón por la que Java Swing esté en desuso.

Sin embargo Swing también tiene otras grandes ventajas. A diferencia que en aplicaciones Web la ejecución de los programas es en las computadoras de los clientes. Esto permite usar toda la potencia de los equipos residentes y no tener que depender de servidores externos que suelen ser más lentos de procesar y muy costosos.

Por todas estas ventajas y por el completo desconocimiento que teníamos hasta el momento en interfaces de escritorio se decidió usar Java Swing para crear la interfaz de nuestra aplicación.

1.1.4. Base de Datos, Oracle

Se eligió usar la base de datos Oracle por la principal razón de usar una base de datos que no se usase a lo larga de la carrera y fuese desconocida para el alumno. Por otro lado, es una base de datos muy potente con una gran cantidad de posibilidades como el soporte multiplataforma o de transacciones entre otras. Oracle posee muchas versiones de las cuales algunas son de pago, en nuestro caso usaremos la versión gratuita "10g Express Edition"

La segunda de las razones que llevaron a elegir Oracle es que es una de las bases de datos más implantadas en este momento.

1.1.5. Prueba de concepto, Préstamo Bibliotecario

Una vez definidas todas las tecnologías que se quieren utilizar, hay que definir qué aplicación se realizará a modo de prueba de concepto.

En la prueba de concepto se creará una aplicación de gestión bibliotecaria que se enfocará en la realización de préstamos de libros por los usuarios/alumnos.

La idea es empezar con la creación de la base de datos en Oracle, que tendrá 3 tablas como base: La tabla "Usuarios", donde se tendrá toda la información referente a los usuarios, la tabla "Libros" que contendrá la información de los libros; y la tabla "Préstamos" que será la encargada de relacionar un libro con un usuario. Gracias a esta tabla podremos crear relaciones N:N entre libros y usuarios, lo que quiere decir que un usuario puede tener varios libros prestados y un libro puede ser prestado a diferentes usuarios en caso de varias copias.

Después mediante Swing se creará la parte de la vista de la aplicación y con la ayuda de Spring conseguiremos que nuestro modelo de clases Java sea completamente desacoplado y fácilmente mantenerle.

Finalmente se usará Hibernate para que la vista realizada en Swing pueda acceder a la información presente en la Base de Datos.

1.2. Objetivos:

El objetivo base del proyecto es investigar y asimilar el funcionamiento del Framework Spring y del ORM Hibernate y hacerlo llegar al mayor número de personas con dichas investigaciones y explicaciones del funcionamiento de ambas tecnologías. Finalmente se realizará una aplicación de escritorio con Swing a modo de prueba de concepto para unir en un solo proyecto las tecnologías desarrolladas en este trabajo de fin de grado. El proyecto se puede dividir diferentes etapas:

- Investigación, análisis y compresión de las diferentes tecnologías en su versión Java que se usaran en el proyecto. Las tecnologías que se usan son las siguientes:
    - Base de datos Oracle albergada en un servidor Apache Tomcat.
    - Framework Spring para gestionar la inyección de dependencias entre clases.
    - Hibernate se encargará de la conexión y comunicación con la base de datos
    - Y finalmente para dotar a nuestro proyecto de una interfaz gráfica dispondremos de la ayuda de Java Swing
- Creación de tutoriales para las principales tecnologías del TFG (Hibernate y Spring), en ellos crearan unos pequeños ejemplos de código con su correspondiente explicación de cómo funcionan Hibernate y Spring por separado.
- Realización de la prueba de concepto donde se unirán todos los conocimientos aprendidos en los procesos anteriores.

1.3. Estructura del documento:

- Capítulo 1, introducción: Aporta la idea general del proyecto, con la motivación para realizarlo, los objetivos fundamentales que pretenden alcanzarse y los medios utilizados para llevar a cabo el proyecto.
- Capítulo 2, estado del Arte: Se explica una visión general de las tecnologías que se utilizaran para la realización del proyecto.
- Capítulo 3, toma de Contacto: Primera toma de contacto con las tecnologías definidas en el proyecto, se crearán unos pequeños ejemplos a modo de tutoriales de las diferentes tecnologías.
- Capítulo 4, prueba de Concepto: Se tratará a fondo la aplicación creada incluyendo casos de uso, captura de requisitos, modelo relacional de la base de datos, modelo relacional de las clases Java y todos los comentarios que se crean oportunos sobre la configuración o código de la aplicación.
- Capítulo 5, futuros trabajos: Se enumeran y expone brevemente propuestas de mejoras o ampliaciones del proyecto existente.
- Capítulo 6, conclusiones: Se expone las conclusiones del alumno al finalizar el proyecto
- Capítulo 7, apéndices: Se detallará un presupuesto de elaboración del proyecto y se mostrará un diagrama de planificación de tareas mediante un diagrama de Gantt.
- Capítulo 8, anexos: Se incluirán todos los anexos que se requieran, principalmente estarán compuestos por partes o clases del código de la prueba de concepto.
- Capítulo 9, bibliografía: Listado de todos los libros, documentos, artículos o páginas Web que se han consultado para la realización del proyecto y la memoria.

1.4. Medios:

A continuación se enumeraran y se dará una breve descripción de los componentes tanto hardware como software que se necesitarán para el desarrollo y puesta en marcha de la aplicación.

Los principales medios Hardware son los siguientes:

- PC con procesador de al menos 2 núcleos y 2 GM de memoria RAM para el correcto funcionamiento del programa y de los programas de desarrollo del mismo.
- Conexión a Internet para consultar posibles dudas o problemas durante el desarrollo.

En cuanto al software necesario:

- Servidor apache Tomcat 7.0 necesario para albergar la Base de Datos.
- Base de Datos Oracle 10g Express Edition.
- Entorno de desarrollo eclipse versión Helios o superior.
- JDK versión 6 para la programación en Java.
- JRE versión 6 para la ejecución de la aplicación.
- Diferentes librerías Java para la programación con las tecnologías que se mostrarán más adelante.

1.5. Glosario de Términos

- JVM o Java Virtual Machine: es un programa ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones generadas por un compilador del lenguaje Java.
- JRE o Java Runtime Envionment: Paquete Java que incluye diversas bibliotecas propias y la JVM. Es completamente necesario tener el JRE instalado en una computadora para ejecutar un programa Java.
- JDK o Java Development Kit: Es el nombre que recibe el kit de desarrollo Java. Contiene todo lo necesario para desarrollar programas en Java entre los que se incluyen un JRE y un compilador e interprete Java, Javadoc, …
- Eclipse: Es un programa de código abierto que se usa para el desarrollo de aplicaciones, soporta gran cantidad de lenguajes, entre ellos Java.
- SQL: Se trata del lenguaje estándar empleando para elaborar las sentencias query o sentencias de acceso a las BBDD.
- Hardware: Son la parte física de un sistema informático
- Software: Son la parte lógica de un sistema informático, está integrada por el Sistema Operativo, los drivers del equipo, los programas,….
- Framework: En español conocido como “marco de trabajo”. Son el conjunto estandarizado de reglas y conceptos enfocados en resolver diferentes problemas programáticos.
- Combo box: También llamado lista desplegable, es un componente usado para mostrar una lista de valores precargados de tal forma que el usuario pueda seleccionar uno de ellos.
- MVC: Es un patrón de programación por el cual se separan los componentes en Modelo (lugar donde se encuentran los datos), Vista (la interfaz que se muestra al usuario) y Controlador (encargado de cargar la información del modelo en la vista y de ejecutar diferentes tareas algorítmicas).

- ORM (Mapeo Objeto-Relacional): es una técnica de programación utilizada por los lenguajes orientados a objetos en la cual se convierten datos de una base de datos relacional a una base de datos orientada a objetos creada virtualmente.
- Framework: Conocido en castellano por el nombre de “marco de trabajo”, son un conjunto de criterios y conceptos estandarizados que sirven para resolver un problema en particular.
- Java Bean: Modelo de componentes para Java. Se usa para encapsular varios objetos en uno solo.
- Beans: También conocidos como Enterprise JavaBeans, específica como los servidores de aplicaciones proveen los objetos del servidor al cliente. El funcionamiento de Spring está basado en ellos.

2. ESTADO DEL ARTE

En este apartado se dará una visión global de las principales tecnologías que se usan en el proyecto.

Se explicará brevemente su historia y la evolución a lo largo del tiempo. Otro de los puntos a tratar serán las principales características de la tecnología y los datos más importantes sobre su estructura o arquitectura.

2.1. Java:

Java es un lenguaje de programación orientado a objetos desarrollado originalmente por "James Gosling" de la empresa "Sun Microsystems", su primera versión fue publicada en el 1995. La sintaxis de Java tiene grandes similitudes con el lenguaje C++, pero a diferencia de este, Java no da tantas facilidades a la hora de programar a bajo nivel. Las aplicaciones Java se compilan una vez y se puede ejecutar en cualquier máquina virtual Java (JVM) sin importar la arquitectura o el sistema operativo del computador.

Java fue proyectado con la finalidad de obtener un producto de pequeñas dimensiones, simple y portátil. Se pretendía que el lenguaje de programación Java fuese orientado a objetos, basado en clases y concurrente. Su lema es "*write once, run anywhere*" lo que quiere decir que los programadores puedes escribir el código una vez y ejecutarlo en cualquier dispositivo sin necesidad de recopilación para otras plataformas

En 2012 java se convirtió en uno de los lenguajes con más uso llegando a más de 10 millones de usuarios gracias a tecnologías propias como las aplicaciones cliente-servidor de web.

2.1.1. Características

El gran cambio que introdujo Java es que es un lenguaje compilado e interpretado. Todo programa en Java se han de compilar al menos una vez y el código bytecodes generado es interpretado por una máquina virtual. De esta forma se independizan máquina y código. El código compilado se ejecuta en máquinas virtuales, que estas si serán susceptibles a la plataforma.

Las principales características de la programación en Java son la portabilidad, el dinamismo, la eficiencia y la seguridad que incorpora. A continuación detallaremos cada una de ellas.

- Se dice que el código Java es portable, ya que es posible ejecutar el mismo archivo de clase, los llamados archivos "*.class", sobre una amplia variedad de arquitecturas de hardware y de software, sin ninguna modificación. Solo es necesaria la existencia de una máquina virtual para el hardware o software elegido para que el programa sea ejecutado.
- Java es un lenguaje dinámico, debido a que las clases son cargadas (desde un sistema de archivos local o desde la red) en el momento en que de necesitan y no antes.
- La eficiencia del lenguaje Java viene dada por el aumento del tipo de datos dinámicamente. Esta característica se la conoce como "late-binding" o enlace tardío y aumenta el tamaño cuando el programa lo necesita. De esta forma se evita el "early-binding" o enlace temprano usado en C o C++ que obliga a asignar memoria y recursos antes de la ejecución del programa. Por lo que de esta forma los programas Java usan menos recursos que los programados en otros lenguajes de programación.
- Java nació cuando ya existía internet así que se diseñó para que fuese seguro y fiable. Se crearon varias capas de seguridad para evitar que programas maliciosos afectaran a los sistemas.

Hasta la aparición de la tecnología JIT (Just In Time) en 1998 a Java se le consideraba un lenguaje lento al tener que interpretar los bytecodes a código nativo

antes de poder ejecutar el programa. Pero con la incorporación del JIT en las máquinas virtuales la conversión de código solo se realizaba una vez y el código nativo resultante se almacenaba para la siguiente ejecución.

2.1.2. Historia

En 1991 la empresa "Sun Microsystems", reclutó un equipo de trabajo al mando de *James Gosling y Mike Sheridan. S*u objetivo era crear un lenguaje para televisión interactiva, pero resultó ser demasiado avanzado para aquel momento. El proyecto terminó derivando en lenguaje de programación Oak que paso a llamarse Green y finalmente Java como lo conocemos en la actualidad.

La procedencia del nombre de Java tiene muchas hipótesis, pero la que más fuerza tiene es que se debe a un tipo de café cercano y de ahí la conclusión de que el símbolo de Java sea una taza de café caliente.

En un principio se utilizó C++ para el proyecto, pero por incompatibilidades se decidió crear un lenguaje propio que sería el Oak mencionado anteriormente, el cual tenía similitudes con C y C++ pero no estaba ligado a un tipo de CPU concreta

A mediados de 1992 el proyecto fue abandonado, pero en 1994 con el nacimiento de la WEB el proyecto se reabrió de tal forma que se enfocó para convertir a Internet en un medio interactivo. En 1994 nació HotJava que fue el primer navegador Web capaz de ejecutar los applet o componentes Java y proporcionaba multiplataforma. En 1995 durante una conferencia se anunció la versión alpha de Java y que Netscape, el navegador más usado en el momento, suportaría Java.

En enero de 1996 se anunció la salida de la primera versión de Java conocida como JDK1.0. SE lanzó con el propósito de "*write once, run anywhere*" comentado anteriormente y pronto todos los navegadores Web de la época incorporaron las appents Java a sus propios navegadores.

Una de las razones por las que Java terminó siendo tan famoso fue la decisión de que el código fuese abierto y gratuito, de tal forma que otras empresas podían crear sus propias tecnologías basadas en Java. Gracias a esto rápidamente los permisos de Java contemplaban importantes firmas como: IBM, Microsoft, Symantec, Silicon Graphics, Oracle, Toshiba y los applets de java eran soportados por los navegadores más importantes de entonces, Nestcape Navigator 3.0 y Microsoft Internet Explorer 3.0

Desde la primera versión de Java ha habido una enorme cantidad de cambios y nuevas versiones, a continuación citaremos brevemente las más importantes con sus principales características:

- JDK 1.0 (23 de enero de 1996)
    - Definido anteriormente.
- JDK 1.1 (19 de febrero de 1997):
    - Introducción de los JavaBeans que eran capaces de encapsular varios objetos en uno mismo.
    - JDBC, un módulo de integración de Bases de Datos. RMI, que invocaba métodos de forma remota.
- J2SE 1.2 (8 de diciembre de 1998):
    - Se introdujo Reflexión, que es capaz de crear objetos dinámicamente dentro de un programa Java
    - Se incorporó Swing, una interfaz gráfica para escritorio.
    - Las JVM incorporaron JIT que ayudaba a la rápida ejecución del código Java.
- J2SE 1.3 (8 de mayo de 2000):
    - JavaSound incorporó efectos básicos de entrada/salida de sonido.
    - Se sustituyó RMI por CORBA que permite diversos componentes software escritos en diferentes lenguajes y se encuentran en diferentes computadoras trabajen juntos
    - Se incorporó una plataforma de Debugg
- J2SE 1.4 (6 de febrero de 2002):
    - Regulación de las expresiones regulares

  - Encapsulamiento de excepciones
  - API de escritura/lectura de imágenes
  - API especializada en logging de aplicaciones
- J2SE 5.0 (30 de septiembre de 2004):
  - Plantillas para conversión de tipos de datos (casting)
  - Se permiten metadatos o anotaciones y se mejora el bucle “for”.

- Java SE 6 (11 de diciembre de 2006):
  - Incluye marcos de trabajo capaces de cambiar de lenguaje Java a otro como PHP o JavaScript.
  - Mejoras de interfaz gráfica y rendimiento
  - Cliente de Servicio Web completo que soporta JAX-WS 2.0, JAXB 2.0, STAX y JAXP.
- Java SE 7 (julio de 2011)
  - Soporta XML dentro del lenguaje Java
  - Soporte para closures o clausuras.
  - Introducción de anotaciones para fallos de Software.

2.2. Hibernate:

Hibernate es una herramienta Java que facilita el mapeo de atributos entre cualquier tipo de Base de Datos relacional (modelo relacional) y el modelo de objetos (orientación a objetos) de una aplicación usando archivos XML para declarar las relaciones.

Hibernate es una herramienta ORM (Object Relational Mapping) con un enorme número de seguidores. Es valorado por muchos incluso como solución superior a productos comerciales dentro de su enfoque, siendo una muestra clara de su reputación y soporte la reciente integración dentro del grupo JBoss que seguramente generará iniciativas muy interesantes para el uso de Hibernate dentro de este servidor de aplicaciones.

Hibernate parte de una filosofía de mapear objetos Java "normales", también conocidos en la comunidad como "POJOs" (Plain Old Java Objects), no contempla la posibilidad de automatizar directamente la persistencia de Entity Beans tipo BMP (es decir, generar automáticamente este tipo de objetos), aunque aun así es posible combinar Hibernate con este tipo de beans utilizando los conocidos patrones para la delegación de persistencia en POJOs.

Una característica de la filosofía de diseño de Hibernate ha de ser destacada especialmente, dada su gran importancia: puede utilizar los objetos Java definidos por el usuario tal cual. Utiliza el mecanismo de reflexión de Java, característica que le permite un modelado iterativo fluido y natural basado en UML, un factor fundamental para lograr un trabajo ágil y productivo. Además abre las puertas a utilizar herencia en el modelo de datos.

2.2.1. Arquitectura

El API de Hibernate es una arquitectura de dos capas (Capa de persistencia y Capa de Negocio).

En la siguiente Figura se muestran los roles de las interfaces Hibernate más importantes en las capas de persistencia y de negocio de una aplicación J2EE. La capa de negocio está situada sobre la capa de persistencia, debido a que actúa como un cliente de la capa de persistencia.

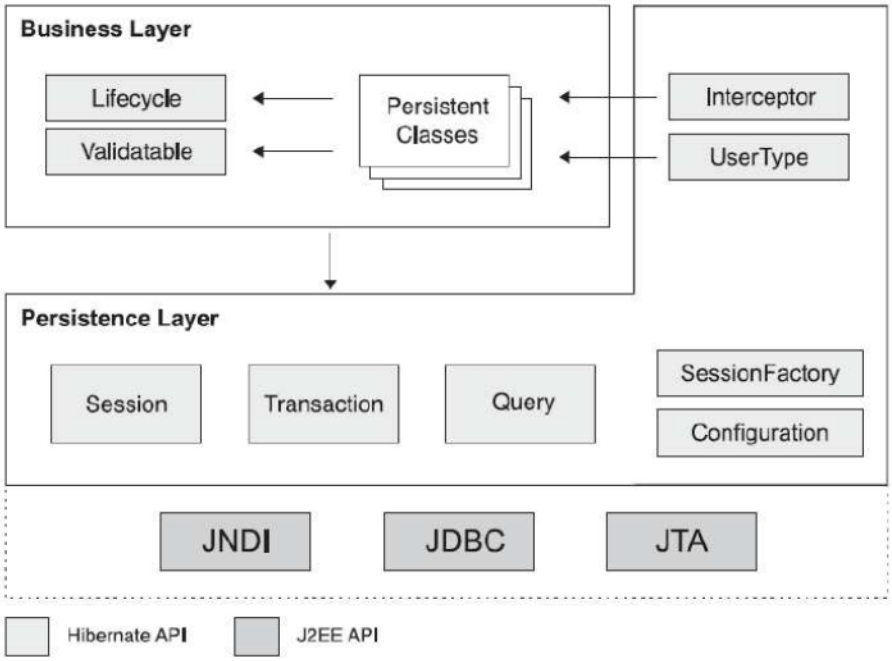


Ilustración 1. Arquitectura de Hibernate

Las Interfaces mostradas se clasifican de la siguiente forma:

- Interfaces llamadas por la aplicación para realizar operaciones básicas:
    - Session: interfaz primaria utilizada en cualquier aplicación Hibernate (SessionFactory).
    - Transaction.
    - Query: permite realizar peticiones a la base de datos y controla cómo se ejecuta dicha petición (query). Las peticiones se escriben en HQL o en el dialecto SQL nativo de la base de datos que estamos utilizando. Una instancia Query se utiliza para enlazar los parámetros de la petición, limitar el número de resultados devueltos por la petición y para ejecutar dicha petición.
- Interfaces llamadas por el código de la infraestructura de la aplicación para configurar Hibernate. La más importante es la clase “Configuration”, se utiliza para configurar y "arrancar" Hibernate. La aplicación utiliza una instancia de

esta para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate, y a continuación crea la Session Factory.

- Interfaces callback que permiten a la aplicación reaccionar ante determinados eventos que ocurren dentro de la aplicación, tales como Interceptor, Lifecycle, y Validatable.
- Interfaces que permiten extender las funcionalidades de mapeado de Hibernate, como por ejemplo UserType, CompositeUserType, e IdentifierGenerator.

Además, Hibernate hace uso de APIs de Java, tales como JDBC, JTA (Java Transaction Api) y JNDI (Java Naming Directory Interface).

2.2.2. Características

Las principales características de Hibernate son las siguientes:

- Programación natural. No requiere de nueva sintaxis ni clases, se puede usar en él cualquier herramienta de programación orientada a objetos.
- No intrusivo (estilo POJO)
- Comunidad activa con muchos usuarios. Muy buena documentación, gran cantidad de libros y foros de consulta
- Posibilidad de usar transacciones, caché, asociaciones, polimorfismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching.
- Potente lenguaje de consulta (HQL) otorgando la posibilidad de crear subqueries, outer joins, ordering, proyeccion (report query) o paginación.
- Fácil testeo.
- Alto rendimiento. Hibernate no necesita tablas extra en Base de Datos y genera gran parte del código SQL al iniciar y no cuando pide información la aplicación
- Persistencia transparente. No requiere de clases extra para la persistencia de datos.

2.2.3. Historia

Hibernate tuvo comenzó su andadura en el año 2001 de manos de un grupo de desarrolladores entre los que destaca Gavin King. Por aquel entonces existían ya los EJB2 (Enterprise JavaBean), por fuentes de programadores de la época se sabe que "era un infierno programar con EJB2" y nunca llegaron a dar el soporte que proponían en un principio. Hibernate nació con la idea de dar las capacidades de persistencia ofrecidas de EJB2 intentando simplificar y añadir nuevas características de las ofrecidas por éste.

En 2003 se comienza el desarrollo de Hibernate2, el cual ofrece muchas mejoras respecto a su primera versión. Tiempo después, JBoss Inc. contrató a los principales desarrolladores de Hibernate y trabajó con ellos en brindar soporte al proyecto.

La rama actual de desarrollo de Hibernate es la 3.x, la cual incorpora nuevas características, como una nueva arquitectura Interceptor/Callback o filtros definidos por el usuario entre otros.

2.3. Spring:

Spring es un framework Java/J2EE para el desarrollo de aplicaciones y contenedor de inversión de control. El código es OpenSource.

Se basa en una configuración a base de JavaBeans. Es potente en cuanto a la gestión del ciclo de vida de los componentes y fácilmente ampliable. Contiene plantillas que facilitan la programación con otras herramientas Java como Hibernate, iBatis, JDBC, JPA... Es interesante el uso de programación orientada a aspectos (IoC).

El objetivo de Spring es ser no intrusivo. Las aplicaciones que lo incorporan no dependen de interfaces o clases de Spring, pero obtienen su configuración a través de las propiedades de sus beans. Este concepto puede ser aplicado a cualquier entorno, desde una aplicación J2EE a un applet.

2.3.1. Arquitectura

La parte fundamental del framework es el módulo Core o "Núcleo", este provee toda la funcionalidad de Inyección de Dependencias (se describirá más adelante) permitiéndole administrar la funcionalidad del contenedor de beans. El concepto básico de este módulo es el BeanFactory, que elimina la necesidad de crear singletons programáticamente permitiéndole desligar la configuración y especificación de las dependencias de la lógica de programación.

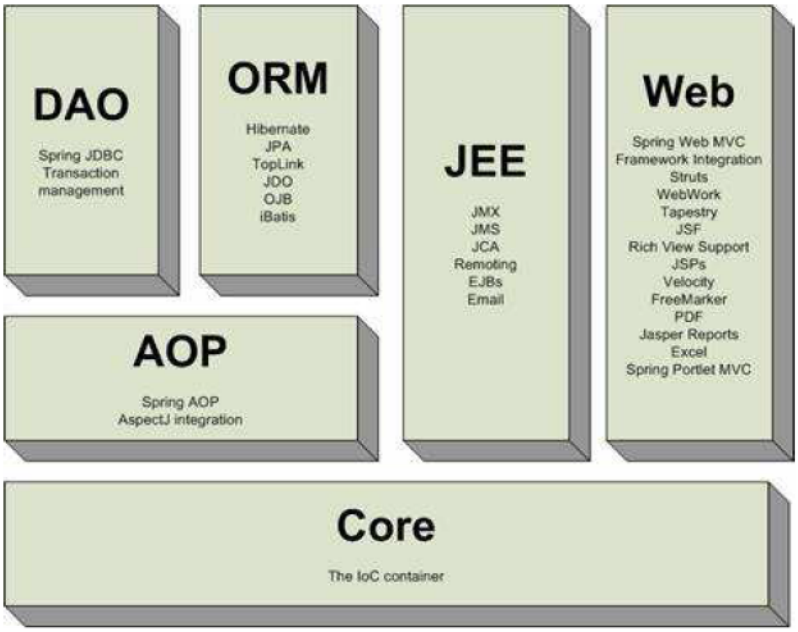


Ilustración 2. Arquitectura Spring

Encima del módulo **core** se encuentra el módulo Context (Contexto), el cual provee de herramientas para acceder a los beans de una manera elegante, similar a un registro JNDI.

El paquete **DAO** provee una capa de abstracción de JDBC que elimina la necesidad de teclear código JDBC redundante y tedioso, así como el manejo de errores de la Base de Datos. También provee de una manera de administrar transacciones tanto declarativas como programáticas.

El paquete **ORM** provee capas de integración para APIs de mapeo objeto - relacional, incluyendo, JDO, Hibernate e iBatis. Usando el paquete ORM es posible usar esos mapeadores en conjunto con otras características que Spring ofrece, como la administración de transacciones mencionada con anterioridad.

El paquete **AOP** provee de programación orientada a aspectos compatible con AOP Alliance, permitiendo definir pointcuts e interceptores de métodos para desacoplar el código de una manera limpia

El paquete **Web** provee características básicas de integración orientadas a la programación web, como realizar la carga de archivos o inicialización de contextos

mediante servlet. También da facilidades para la inclusión de herramientas como WebWork o Struts.

El paquete **Web MVC** provee de una implementación Modelo - Vista - Controlador para las aplicaciones web. La implementación de Spring MVC permite una separación entre código de modelo de dominio y las formas web y permite el uso de otras características de Spring Framework como lo es la validación.

2.3.2. Características

Las principales características de Spring son las siguientes:

- Complejidad muy reducida en comparación con otros frameworks de su tipo. Configuración de las aplicaciones rápida y sencilla.
- Contiene capacidades avanzadas para la escalabilidad de la complejidad de los programas.
- Posee una capa abstracción de JDBC que simplifica el manejo de errores.
- Facilidades para usar MVC directamente montado sobre Spring
- Capacidad de integrar y configurar otros tipos de herramientas.

2.3.3. Historia

Spring dio sus primeros pasos en el año 2.000 de la mano de un grupo de expertos desarrolladores Java entre los que destacan Rod Johnson y Jürgen Höller. Su principal objetivo fue simplificar partes de la plataforma J2EE que se usaba por aquel entonces.

En 2.003 se formó un equipo de desarrollo para el framework y un año después se lanzó su primera versión ganando popularidad rápidamente gracias a ser de código abierto y contener una documentación Javadoc completa.

Rápidamente Spring fue desbancando a los JavaBean que según palabras de Walls el creador de Spring decía que:" EJB es complicado porque EJB fue creado para resolver cosas complicadas".

Durante los siguientes años Spring fue acaparando más popularidad gracias a sus nuevas versiones que fueron incluyendo nuevos módulos como la inversión de control o la programación orientada a aspectos.

3. TOMA DE CONTACTO TECNOLOGÍAS

3.1. Hibernate:

Hibernate es un marco de trabajo del tipo ORM (Object Relational Mapping) "Mapeo Objeto Relacional". Se encarga de todos los detalles de trabajar con las bases de datos y genera el código SQL por sí solo, es necesario indicarle cómo hacerlo con diferentes clases y ficheros XML de configuración.

El manejo de Hibernate puede resultar muy complicado si no se cuenta con la documentación adecuada, aquí se mostrará un ejemplo básico de cómo usar este marco de trabajo. Para Facilitar el trabajo de aprendizaje se entrega junto a la este documento un proyecto java con el ejemplo en que se basa este tutorial. En el ejemplo partimos de la tabla "Usuarios" que tenemos en nuestra base de datos y a partir de ahí crearemos todo lo necesario para utilizar Hibernate sobre ella.

3.1.1. Hibernate.cnf.xml

La base de la estructura de Hibernate es el fichero **hibernate.cnf.xml** al cual le tendremos que indicar los datos de la conexión a la base de datos, la localización de las clases de persistencia y opcionalmente otros datos de configuración.

Como todo fichero XML comienza con 2 encabezados y a continuación se crea una etiquita sesión-factory, **<sesion-factory>**, dentro de la cual se definen diferentes property los cuales definiremos a continuación:

```
<property name="dialect"> Definir la base de datos que
usaremos para que sepa el dialecto SQL a usar
<property name="connection.url"> Definir la conexión a la
BD
<property name="connection.username"> Usuario de la BD
<property   name="connection.password">   Contraseña   del
usuario de BD
<property  name="connection.driver_class"> Definir  el  tipo
```

```
de driver a utilizar
<property     name="myeclipse.connection.profile">     Definir
perfil creado para interactuar con la BD
```

Una vez definidos todos estos parámetros tan solo nos queda por indicarle a Hibernate la información de la infraestructura de datos, en este caso se realizara creando una etiqueta **<mapping>** indicando ruta y nombre de clases de persistencia pertenecientes a cada una de las tablas que usaremos en el proyecto. Por lo que tendremos que crear tantas etiquetas mapping como tablas de la base de datos queramos usar

Para facilitar el comprendimiento de la estructura y datos de este fichero, en el anexo ANEXO1 encontraremos un ejemplo de fichero de configuración **hibernate.cnf.xml** en el que se usa Oracle como base de datos.

3.1.2. Tablas, Objetos y Clases de Persistencia

La finalidad de Hibernate es que podamos tratar como colecciones de objetos los datos que tenemos en nuestra base de datos.

Para realizar esta conversión lo primero que necesitamos es tener creada una tabla en la base de datos y un objeto Java que simule dicha tabla con tipos de dato similares. De tal forma que si tenemos en nuestra base de datos una tabla llamada "Usuarios" con un campo llamado "nombre" del tipo varchar2(80), crearemos un objeto Usuarios (es aconsejable que el nombre de la tabla y el objeto coincidan) con un atributo de tipo String con el mismo nombre.

Una vez tenemos la tabla de la base de datos y el objeto homónimo en Java tenemos que crear una clase de persistencia mediante un fichero XML en el que dentro de una etiqueta **<hibernate-mapping>** iremos añadiendo las diferentes conexiones de la siguiente forma:

```
<class name="com.rivas.hibernate.Usuarios" table="USUARIOS"
       schema="RIVAS">
```

```
    <property name="nombre" type="java.lang.String">
        <column name="NOMBRE" length="80" />
    </property>
...
```

De esta forma le indicamos a Hibernate que la tabla “Usuarios” y la clase presente en la ruta “com.rivas.hibernate.Usuarios” son las elegidas para crear la persistencia. De forma similar a la que mostramos con la variable “nombre” que es una cadena de caracteres, podemos crear la persistencia de cualquier tipo de dato como puede ser datos numéricos, fechas, etc.

En el anexo ANEXO2 tenemos un ejemplo completo de cómo crear una clase de persistencia de una tabla.

Como último apunte recordar que cada vez que creamos una nueva clase de persistencia tenemos que indicárselo a Hibernate en el fichero **hibernate.cnf.xml** mediante una nueva etiqueta mapping con el nombre y la ruta de la clase de persistencia.

3.1.3. HibernatesessionFactory.java

Para poder usar Hibernate necesitamos que nuestros DAO (de los cuales hablaremos más adelante) tengan acceso a las sesiones que crea Hibernate llamadas sessionFactory, de esto se encargará la clase “HibernatesessionFactory”. No profundizaremos en el funcionamiento de esta clase ya que la mayoría de los proyectos en Hibernate tienen formas muy similares de acceder al sessionFactory.

Tan solo hay un parámetro configurable en la clase y es la ruta de nuestro fichero de configuración hibernate.cnf.xml, la cual tendremos que indicar para que Hibernate funcione correctamente.

Crearemos un método llamado getSession() el cual se encargará de abrir o crear el sessionFactory necesario y devolverlo en un tipo de dato Session al DAO que nos lo solicite. Una vez el DAO correspondiente tiene la sesión podrá realizar query’s a la base de datos mediante el lenguaje HSQL

En el anexo ANEXO3 se facilita un ejemplo completo de cómo crear una clase HibernatesessionFactory.

3.1.4. Patrón DAO

En este momento tenemos nuestro Hibernate configurado, la clase de persistencia para enlazar tabla y objetos Usuarios y también tenemos la forma de acceder al HibernatesessionFactory pero ¿Cómo hacemos nuestras query's a la base de datos? En este momento es cuando entra en juego el patrón DAO usado en Hibernate, que contiene las órdenes de acceso a la base de datos.

En nuestro caso crearemos una clase que llamaremos "UsuariosDAO.java", que será la encargada de usar la clase de persistencia e interactuar entre nuestro programa y la base de datos. Usando la interfaz HibernateSessionFactory la cual contiene numerosos métodos con los cuales acceder a nuestras tablas.

Dentro de esta clase se crearan tantos métodos como query's diferentes queramos realizar. A continuación mostraremos el código de cómo realizar una sentencia SELECT mediante Hibernate:

```
public List findAll() {
	try {
		String queryString = "from Usuarios";
		Query queryObject =
HibernateSessionFactory.getSession().createQuery(queryString);
		return queryObject.list();

	} catch (RuntimeException re) {
		throw re;
	}
}
```

En el método mediante "HibernateSessionFactory.getSession()" se pide una sesión de Hibermante y con ".createQuery(queryString);" le indica que tipo de sentencia se quiere realizar. Cuando tiene toda la lista de resultados devuelve en un tipo list de java la lista completa de la sentencia realizada.

Usando el "`HibernateSessionFactory.getSession()`" de la misma forma vista antes, se pueden realizar otro tipo de consultas como pueden ser el salvado:

```
public void save(Usuarios Mi_Usuario) {
        HibernateSessionFactory.getSession().save(Mi_Usuario);
        …
```

El borrado:

```
public void delete (Usuarios Mi_Usuario) {
        HibernateSessionFactory.getSession().delete(Mi_Usuario);
        …
```

Y otras muchos tipos de sentencias que veremos en nuestra prueba de concepto

3.1.5. Ejemplo Final

Una vez tenemos montado todo nuestro sistema de Hibernate tan solo nos falta hacer un ejemplo que lo use. Se pretende mostrar mediante un Combo Box (lista desplegable) la realización de una sentencia SELECT contra la tabla "Usuarios"

Para ello se crea una ventana con swing en la que creamos una instancia de la clase UsuariosDAO.java y mediante esta instancia llamamos al método "findAll()" definido anteriormente, nos devuelve una lista de objetos Usuario con todos los usuarios presentes en la tabla.

```
UsuariosDAO UsuDAO = new UsuariosDAO();
List<Usuarios> lista_usuarios=  UsuDAO.findAll();
```

Para finalizar, seleccionaremos los campos de los objetos que queramos mostrar y le pasaremos la lista al objeto JComboBox para que la muestre.

A continuación se muestra una imagen del ejemplo completado en el que mostramos el nombre y apellido de nuestros Usuarios de la base de datos.

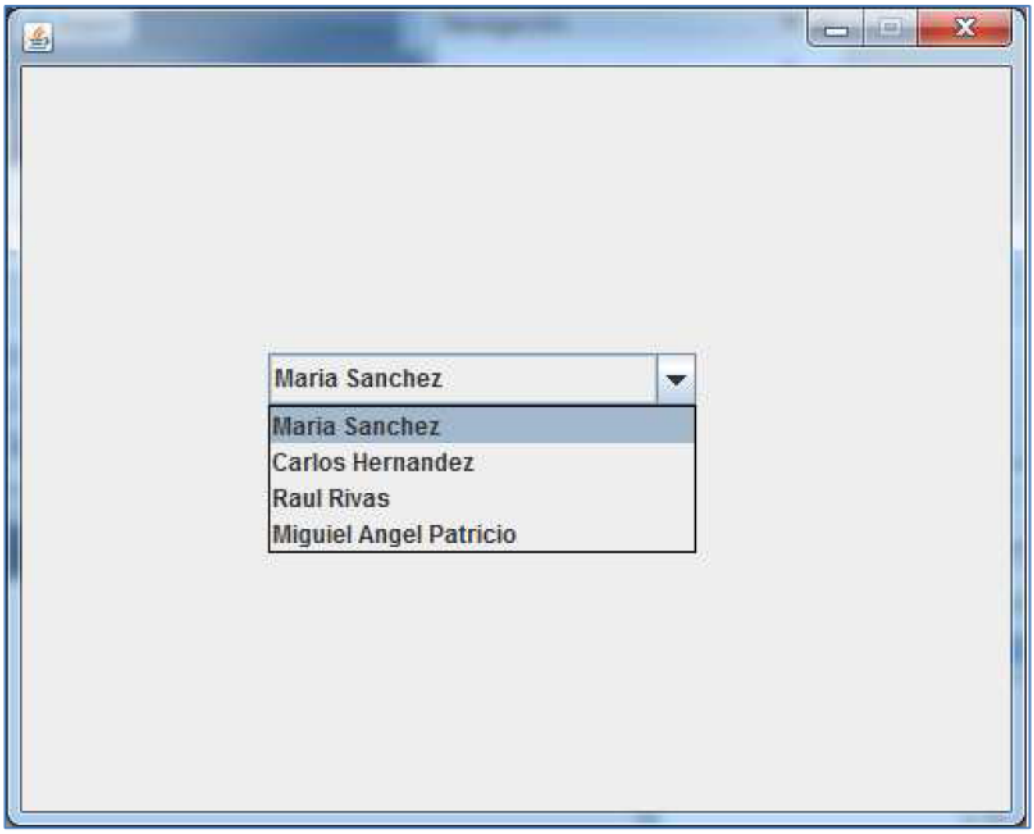


Ilustración 3. Ejemplo Hibernate

3.2. Spring:

Spring es un framework que se usa para conseguir desacoplamiento entre clases Java implementando dos patrones; el patrón de DI (inyección de dependencias) o inversión de control (IOC). La misión de Spring es actuar de intermediario entre las clases por eso también se le dice que es un middleware

El framework de Spring es enorme, por eso nosotros nos centraremos en la parte que nos interesa que es la inyección de dependencias. Lo que buscamos con Spring es tener dependencias débiles entre clases de tal forma que el mantenimiento de la aplicación se facilita enormemente.

Al igual que antes, para Facilitar el trabajo de aprendizaje se entrega junto a la memoria un proyecto Java con el ejemplo en que se basa este tutorial. Haremos un ejemplo muy simple en el cual tendremos dos clases. La primera de estas llamará a un método perteneciente a la segunda mediante Spring con lo que la primera clase no crea una dependencia, es el framework Spring quien le entrega dicha dependencia.

3.2.1. Aplication context.xml

Spring necesita un fichero descriptor en el que se le indique de que clases tendrá que crearse dependencias, en nuestro caso usaremos el fichero AplicationContext.xml. Antes de seguir hemos de saber que para usar Spring es necesario que las clases a inyectar sean “Beans”.

Como cualquier fichero XML, tendrá unas cabeceras y luego definiremos los Beans a utilizar en la aplicación y la ruta de la clase Acceso_AplicationContext.java que es la clase que interactuará directamente con el XML para crear los Beans.

Para definir cada uno de los Beans que querremos inyectar con Spring crearemos una nueva etiqueta Beans en el xml de la siguiente forma:

```
<bean id="prueba"
      class="com.rivas.controlador.Prueba" scope="prototype" />
```

En la etiqueta hemos de indicarle el id del Beans, la ruta en la que se encuentra y el valor de scope que en nuestro caso lo hemos puesto a prototype con lo que el Beans creado se instanciará todas las veces que sea necesario.

Dentro de las cabeceras del AplicationContext.xml se decidió que el enlazado será por nombre "`default-autowire="byName"` "con lo que el campo id es necesario que sea único.

El otro dato importante a indicarle al AplicationContext es el lugar donde se encuentra el Acceso_ApplicationContext.java que será el encargado de interactuar y entre el programa principal y el AplicationContext.xml.

```
<bean id="acceso_contextos"class="com.rivas.spring.Acceso_ApplicationContext"
            scope="singleton" lazy-init="false" />
```

En este caso el valor scope del Beans está a singleton, de esta forma le indicamos que tan solo queremos crear este Beans una vez, que normalmente se realizará al iniciar el programa principal.

En el anexo ANEXO4 tenemos un ejemplo completo de cómo quedaría nuestro AplicationContext.xml listo para funcionar.

3.2.2. Acceso Aplication context

Como ya se dijo antes la clase Acceso_ApplicationContext.java será la encargada de interactuar entre el xml y el programa principal, será el encargado de obtener los Beans creados por Spring y entregárselos al programa principal.

En esta clase se tendrá un objeto del tipo AplicationContext "`private static ApplicationContext ctx;`" del cual tendremos sus métodos accesores, método get y set.

La parte que más nos interesa de la clase es el método getBean:

```
public static Object getBean(String objeto) {
	return ctx.getBean(objeto);
}
```

Con el que crearemos los Beans que se nos pidan. El método recibe por parámetro un String que será el ID del Beans a crear (en nuestro caso el String será "prueba" ya que solo tenemos un Beans definido en el XML) y devuelve el objeto del mismo.

Para resolver cualquier posible duda en el anexo ANEXO5 tenemos un ejemplo de la clase Acceso_ApplicationContext.java completa.

3.2.3. Clases e Interfaces del programa

Ya tenemos los ficheros necesarios para utilizar Spring, ahora explicaremos que tenemos que hacer y que clases e interfaces tenemos que crear para usarlo en nuestro programa:

3.2.3.1. Prueba.java:

Lo primero que crearemos es una clase llamada Prueba.java de la que vamos a crear su Beans mediante Spring. Este tipo de clases que vamos a inyectar pueden tener constructor pero siempre tendrá que estar vacío ya que todas las llamadas que hagamos serán a sus métodos, nunca a los constructores

Siguiendo con la clase Prueba.java, crearemos un método, en nuestro caso es un método sumador, recibe 2 enteros y los suma. No nos interesa la complejidad del método sino que Spring lo inyecte correctamente.

3.2.3.2. *Interface_prueba.java:*

Spring a la hora de llamar a las clases para crear sus Beans necesita que estas implementen interfaces por lo que el siguiente paso es crear una interfaz para nuestra clase "Prueba.java" en la que tendremos que definir qué métodos tendrá la clase que la implemente.

3.2.3.3. *Inicio.java:*

Finalmente crearemos una clase con nuestro main desde la cual crearemos gracias a Spring una dependencia con la clase "Prueba.java".

En esta clase tendremos que realizar varias cosas. La primera de ellas será indicarle al programa la ruta de donde se encuentra el fichero de configuración de Spring ApplicationContext.XML mediante el método "ClassPathXmlApplicationContext":

```
ApplicationContext contexto = new ClassPathXmlApplicationContext(
            "com/rivas/spring/applicationContext.xml");
```

El siguiente paso será crear el Beans de la clase prueba en la clase Inicio:

```
Interface_prueba mi_prueba = (Interface_prueba)
Acceso_ApplicationContext.getBean("prueba");
```

Crearemos una interfaz llamada "mi_prueba" que es del tipo Interface_prueba y con ella llamaremos al método getBean del Acceso_ApplicationContext pasándole como parámetro el ID del Beans a crear, en este caso "prueba".

Una vez hecho esto podemos usar la interfaz creada "mi_prueba" para realizar llamadas a los métodos pertenecientes a la clase "Prueba.java".

```
int resultado = mi_prueba.suma(5, 7);
System.out.println(resultado);
```

Con la interfaz mi_prueba creada podemos hacer tantas llamadas como queramos a los métodos de prueba.java y si esta clase tuviera más métodos también podríamos acceder a ellos de forma similar.

Para finalizar les dejamos una captura de pantalla de una ejecución del programa realizada en eclipse y les recordamos que si tienen cualquier duda tiene el ejemplo del proyecto completo en la documentación.

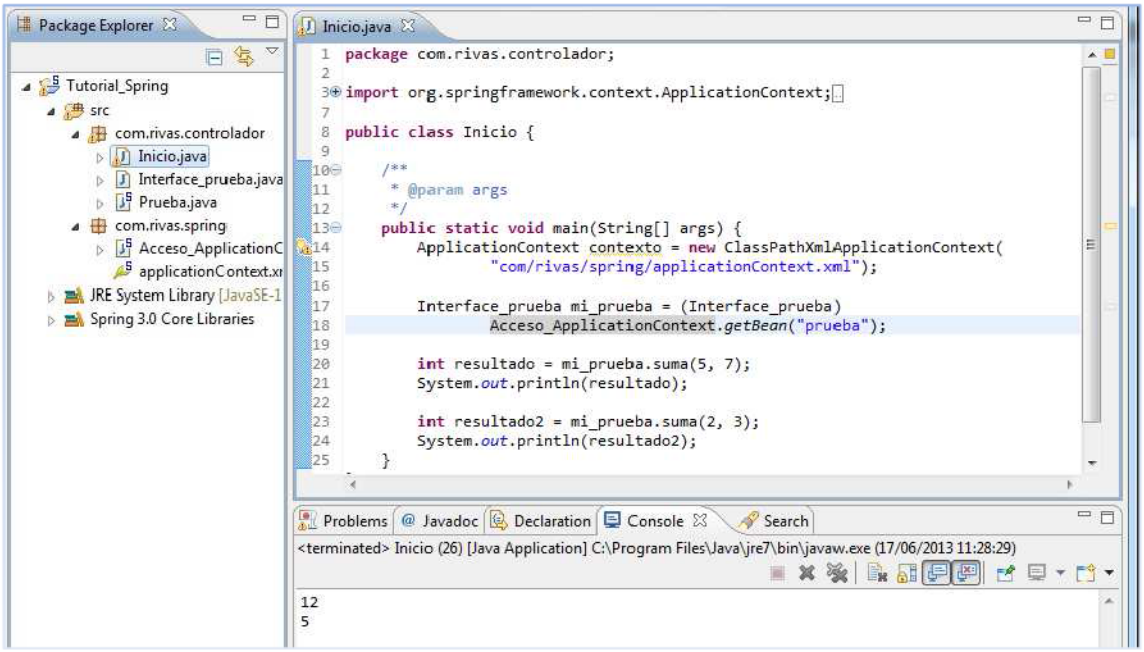


Ilustración 4. Ejemplo Spring

4.PRUEBA DE CONCEPTO

4.1. Arquitectura

Antes de comenzar con la descripción de la aplicación se debe puntualizar que, en la medida de todo lo posible, en la aplicación se seguirá el modelo MVC (Modelo Vista Controlador) cuya finalidad es la de desarrollar aplicaciones mediante tres capas, las cuales definiremos a continuación:

- Modelo: Es la representación específica de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.
- Controlador: Se podría decir que el controlador es el intermediario entre el modelo y la vista. Es el encargado de responder los eventos q le envían tanto la vista como el controlador y actuar en consecuencia a ellos.
- Vista: Es el interfaz del usuario, encargado de mostrar correctamente los datos del modelo de la aplicación.

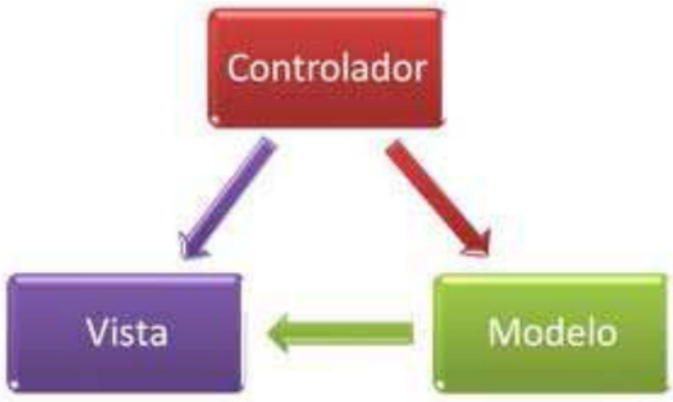


Ilustración 5. MVC

4.2. Especificación de Requisitos

4.2.1. Requisitos Funcionales

Son el conjunto de operaciones que el sistema o la aplicación debe soportar, es decir, lo que debe hacer. Hay veces que es de gran utilidad describir qué operaciones o qué comportamiento no va a tener para no dar lugar a equívocos.

A continuación se hará un listado de los distintos requisitos funcionales de la aplicación que se desarrollará:

- RF-1: La interfaz de la aplicación debe ser sencilla e intuitiva y en lo medida de lo posible amigable. Dispondrá de una ventana principal desde la cual se accederá a las diferentes vistas. No se crearán nuevas ventanas, toda la información se cargará sobre la ventana principal.
- RF-2: El idioma de la aplicación podrá ser cambiado entre español, alemán e inglés.
- RF-3: La aplicación recordará el idioma seleccionado por el usuario para iniciarse en el lenguaje seleccionado por el usuario.
- RF-4: Se permitirá crear nuevos escritores y modificar o eliminar los ya existentes en la aplicación.
- RF-5: La eliminación de los escritores no eliminaran los datos de los mismos, simplemente pasarán a estar “desactivados”.
- RF-6: Se dispondrá de un listado de los escritores existentes en la aplicación que no se encuentren desactivados.
- RF-7: Se permitirá crear nuevos libros y modificar o eliminar los diferentes libros ya almacenados en la aplicación.

- RF-8: La eliminación de los libros no eliminaran los datos de los mismos, simplemente pasarán a estar “desactivados”.
- RF-9: Se dispondrá de un listado de los libros existentes en la aplicación que no se encuentren desactivados.
- RF-10: En la misma vista que el RF9, se dispondrá de un filtrado de libros por escritor, en el cual solo se mostrarán solo los libros asignados al escritor seleccionado.
- RF-11: Se permitirá crear nuevos usuarios y modificar o eliminar los ya existentes en la aplicación.
- RF-12: La eliminación de los usuarios no eliminaran los datos de los mismos, simplemente pasarán a estar “desactivados”.
- RF-13: Se dispondrá de un listado de los usuarios existentes en la aplicación que no se encuentren desactivados.
- RF-14: Se permitirá el préstamo de libros y la devolución de los mismos por parte de los usuarios.
- RF-15: La devolución de un préstamo no eliminará los datos de la aplicación, simplemente se marcará el préstamo como devuelto.
- RF-16: Se dispondrá de un listado de los préstamos existentes en la aplicación que no estén marcados como devueltos.
- RF-17: Cada usuario no podrá tener más de 3 libros prestados a la vez.
- RF-18: Al intentar cerrar la aplicación aparecerá una ventana de diálogo para confirmar la acción de cerrado.

4.2.2. Requisitos no funcionales

Son el conjunto de funciones o especificaciones adicionales que se deben de tener en cuenta a la hora de desarrollar la aplicación y su puesta en funcionamiento, especificaciones tales como el rendimiento del sistema, los interfaces que usará, etc.

Estos son los requisitos no funcionales que aplicaremos al desarrollo de la aplicación:

- RNF-1: El sistema operativo que se utilizará será Windows XP o superior para el correcto funcionamiento de la aplicación.
- RNF-2: El equipo donde se ejecutará la aplicación deberá disponer de al menos 2 GB de memoria RAM para el correcto funcionamiento.
- RNF-3: Será necesario el uso del teclado y del ratón para trabajar con la aplicación.
- RNF-4: Será necesario tener instalado en el equipo el Sistema Gestor de Base de Datos, que en nuestro caso es Oracle XE.
- RNF-5: El equipo donde se aloje la aplicación deberá tener instalada al menos la versión 7 de máquina virtual de Java (JRE).

4.3. Análisis

En este apartado se detallará el análisis de la aplicación a desarrollar. Se dividirá en tres secciones. En la primera, se detallaran los distintos casos de uso con sus correspondientes diagramas.

La segunda parte consistirá en el diagrama de clases Java de la aplicación de alto nivel.

Para terminar se incluirá el modelo entidad relación de alto nivel de la base de datos para el desarrollo del programa.

4.3.1. Casos de Uso

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Para ser más concretos, son las interacciones que la persona que gestione el programa tiene que llevar a cabo con la misma. A este actor le llamaremos administrador dentro de los casos de uso.

A continuación, se exponen los casos de uso de la aplicación incluyendo una descripción detallada de cada uno de ellos. En el caso de que aparezca más de una vez un caso de uso en algún diagrama, solamente se detallará una única vez.

Se dispondrán los diagramas de casos de uso con su correspondiente descripción para obtener así una mejor perspectiva de la funcionalidad de la aplicación. Se han agrupado estos diagramas para facilitar la comprensión de los mismos. De esta forma dividiremos los casos de uso en: casos de uso de escritores, casos de uso de libros, casos de uso de usuarios y casos de uso de préstamos.

4.3.1.1. Casos de uso de escritores

El siguiente diagrama muestra las acciones que puede realizar el administrador sobre los escritores que se encuentran en la Base de Datos:

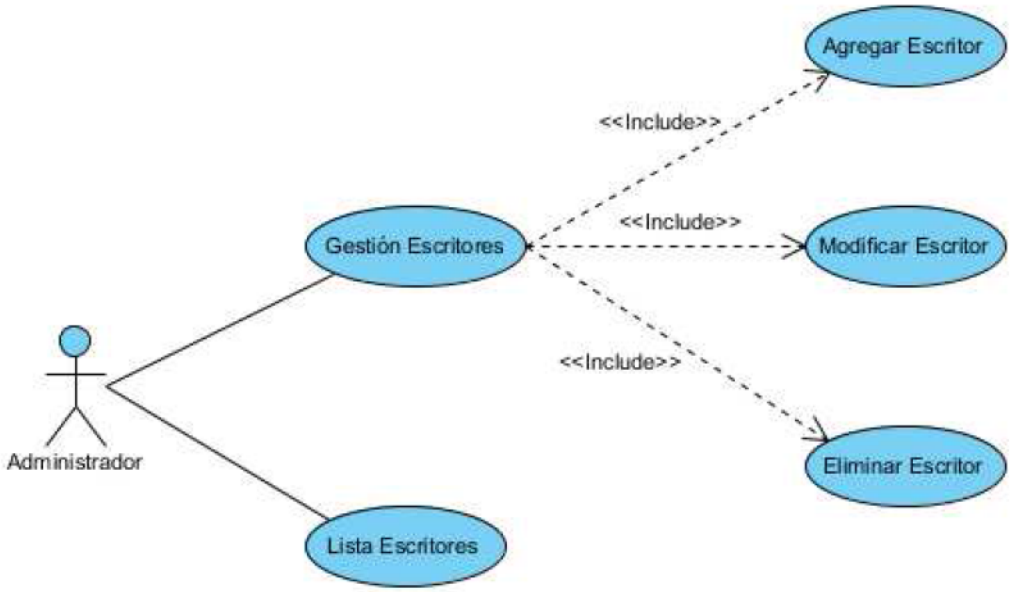


Ilustración 6. Casos de uso escritores

A continuación detallaremos correctamente cada uno de los casos de uso incluidos dentro del diagrama.

Tabla 1. Casos de uso, agregar escritor

Nombre Caso de Uso	**Agregar un escritor nuevo**
Descripción	Crear un nuevo escritor para ser utilizado en la gestión de libros.
Actores	Administrador.
Precondiciones	El escritor no puede estar en la base de datos de la aplicación (se comprobarán las cadenas de nombre + apellido introducidas)
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Escritores" → "Agregar Escritores" para acceder a la ventana de "Agregar Escritor Nuevo". Rellenará todo el formulario y le dará al botón de "guardar".
Excepciones	- Si alguno de los campos del formulario está vacío aparecerá una pantalla informando que faltan campos. - En el caso de que el escritor ya exista el guardado no se llevará a cabo y se avisará mediante una pantalla emergente.
Postcondiciones	- Se guardará el escritor en la base de datos de la aplicación. - Se mostrará una ventana informando de que el guardado se llevó a cabo correctamente.

Tabla 2. Casos de uso, modificar escritor

Nombre Caso de Uso	**Modificar un escritor**
Descripción	Permite modificar la información de los escritores existentes en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un escritor a modificar. - El escritor a modificar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en pantalla.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Escritores" → "Modificar o Eliminar Escritores" para acceder a la ventana de "Modificar o Eliminar Escritor". Al seleccionar un escritor del combo box que aparece, se cargarán los datos del escritor seleccionado en los campos. Una vez modificados los campos se pulsará el botón "Guardar Modificación".
Excepciones	Si alguno de los campos del formulario se modifica a vacío aparecerá una pantalla informando que faltan campos en el formulario.
Postcondiciones	- Se guardará las modificaciones del escritor en la base de datos de la aplicación. - Se mostrará una ventana informando de que la modificación se llevó a cabo correctamente.

Tabla 3. Casos de uso, eliminar escritor

Nombre Caso de Uso	Eliminar un escritor
Descripción	Permite marcar como eliminado un escritor existente en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un escritor a eliminar. - El escritor que se desea eliminar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en la pantalla
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Escritores" → "Modificar o Eliminar Escritores" para acceder a la ventana de "Modificar o Eliminar Escritor". Al seleccionar un escritor del combo box que aparece, se cargarán los datos del escritor seleccionado en los campos y se pulsará el botón "Eliminar Escritor".
Excepciones	No se contemplan
Postcondiciones	- Se guardará el estado de "eliminado" del escritor en la base de datos de la aplicación. - Se limpiarán los campos del formulario y se actualizará la lista de escritores que se muestra en el combo box. - se mostrará una ventana informando de que la eliminación se llevó a cabo correctamente.

Tabla 4. Casos de uso, listado de escritores

Nombre Caso de Uso	Listado de escritores
Descripción	Muestra una lista de los escritores existentes en la aplicación, solo mostrará los escritores que no estén marcados como eliminados.
Actores	Administrador.
Precondiciones	No se contemplan
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Escritores" → "Listado de Escritores" para acceder a la pantalla de "Listado de Escritor".
Excepciones	En caso de no existir ningún escritor la tabla se mostrará vacía.
Postcondiciones	No se contemplan

4.3.1.2. Casos de uso de libros

El siguiente diagrama muestra las acciones que puede realizar el administrador sobre los libros que se encuentran en la Base de Datos:

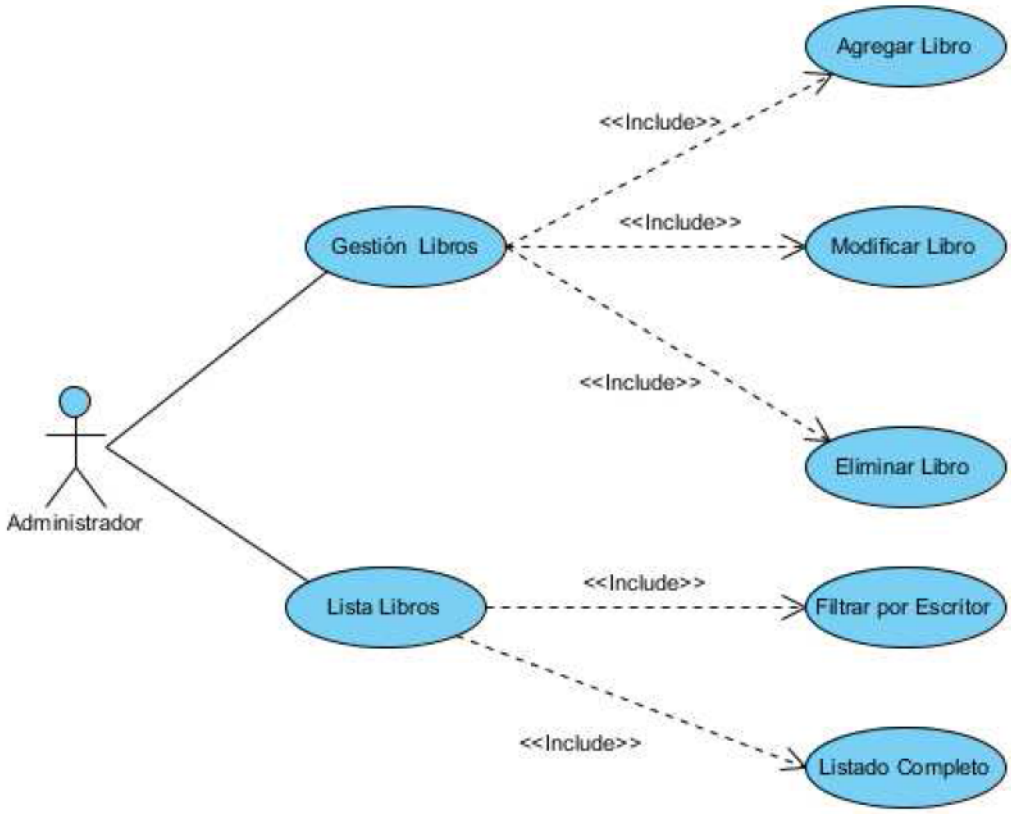


Ilustración 7. Casos de uso libros

A continuación detallaremos correctamente cada uno de los casos de uso incluidos dentro del diagrama.

Tabla 5.Casos de uso, agregar libro

Nombre Caso de Uso	**Agregar un Libro nuevo**
Descripción	Crear un nuevo libro para ser utilizado en la gestión de préstamos.
Actores	Administrador.
Precondiciones	El Libro no puede estar en la base de datos de la aplicación (se comprobarán la cadena del nombre del libro introducido).
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Libros" → "Agregar Libro" para acceder a la ventana de "Agregar Libro Nuevo". Rellenará todo el formulario y le dará al botón de "guardar".
Excepciones	- Si alguno de los campos del formulario está vacío aparecerá una pantalla emergente informando que faltan campos por completar. - El único campo que puede estar vacío es la portada, en caso de que portada este vacío se le asignará una portada por defecto - En el caso de que el libro ya exista el guardado no se llevará a cabo y se avisará mediante una pantalla emergente.
Postcondiciones	- Se guardará el libro en la base de datos de la aplicación. - Se mostrará una ventana informando de que el guardado se llevó a cabo correctamente.

Tabla 6. Casos de uso, modificar libro

Nombre Caso de Uso	**Modificar un libro**
Descripción	Permite modificar la información de los libros existentes en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un libro a modificar. - El libro que se desee modificar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en pantalla.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Libros" → "Modificar o Eliminar Libros" para acceder a la ventana de "Modificar o Eliminar Libro". Al seleccionar un libro del combo box que aparece, se cargarán los datos del libro seleccionado en los campos. Una vez modificados los campos se pulsará el botón "Guardar Modificación".
Excepciones	Si alguno de los campos del formulario se modifica a vacío aparecerá una pantalla informando que faltan campos en el formulario.
Postcondiciones	- Se guardará las modificaciones del libro en la base de datos de la aplicación. - Se mostrará una ventana informando de que la modificación se llevó a cabo correctamente.

Tabla 7.Casos de uso, eliminar libro

Nombre Caso de Uso	**Eliminar un Libro**
Descripción	Permite marcar como eliminado un libro existente en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un libro a eliminar. - El libro que se desee modificar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en pantalla.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Libros" → "Modificar o Eliminar Libros" para acceder a la ventana de "Modificar o Eliminar Libro". Al seleccionar un libro del combo box que aparece, se cargarán los datos del libro seleccionado en los campos y se pulsará el botón "Eliminar Escritor".
Excepciones	No se contemplan
Postcondiciones	- Se guardará el estado de "eliminado" del libro en la base de datos de la aplicación. - Se limpiarán los campos del formulario y se actualizará la lista de libros que se muestra en el combo box. - se mostrará una ventana informando de que la eliminación se llevó a cabo correctamente.

Tabla 8. Casos de uso, listado de libros

Nombre Caso de Uso	Listado de Libros completo
Descripción	Muestra una lista de los libros existentes en la aplicación, solo mostrará los libros que no estén marcados como eliminados.
Actores	Administrador.
Precondiciones	No se contemplan
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Libros" → "Listado de Libros" para acceder a la pantalla de "Listado de Libros".
Excepciones	En caso de no existir ningún libro la tabla se mostrará vacía.
Postcondiciones	No se contemplan

Tabla 9.Casos de uso, listado de libros con filtrado

Nombre Caso de Uso	Listado de Libros con filtrado
Descripción	Muestra una lista de los libros existentes en la aplicación, solo mostrará los libros que no estén marcados como eliminados y tan solo los pertenecientes al escritor por el que se desea filtrar.
Actores	Administrador.
Precondiciones	No se contemplan
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Libros" → "Listado de Libros" para acceder a la pantalla de "Listado de Libros". Al seleccionar un escritor del combo box que aparece, se cargarán en la tabla los datos de todos los libro pertenecientes al escritor elegido.
Excepciones	En caso de no existir ningún libro del autor seleccionado, la tabla se mostrará vacía.
Postcondiciones	Se mostrará el listado de libros del escritor seleccionado

4.3.1.3. Casos de uso de usuarios

El siguiente diagrama muestra las acciones que puede realizar el administrador sobre los usuarios que se encuentran en la Base de Datos:

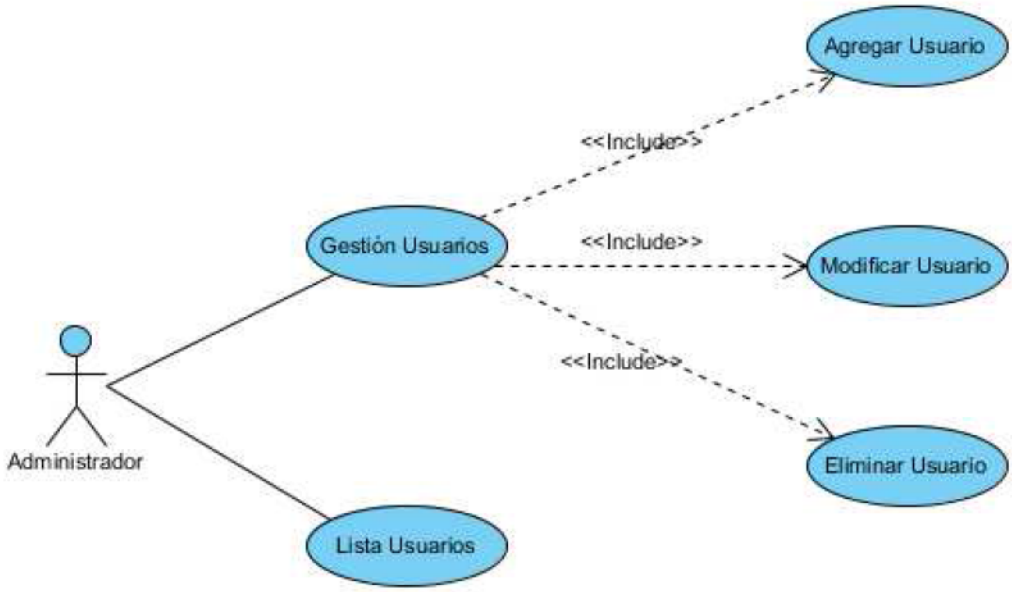


Ilustración 8. Casos de uso usuarios

A continuación detallaremos correctamente cada uno de los casos de uso incluidos dentro del diagrama.

Tabla 10.Casos de uso, agregar usuario

Nombre Caso de Uso	**Agregar un usuario nuevo**
Descripción	Crear un nuevo usuario para ser utilizado en la gestión de préstamos.
Actores	Administrador.
Precondiciones	El usuario no puede estar en la base de datos de la aplicación (se comprobarán las cadenas de nombre + apellido introducidas).
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Usuarios" → "Agregar Usuarios" para acceder a la ventana de "Agregar Usuario Nuevo". Rellenará todo el formulario y le dará al botón de "guardar".
Excepciones	- Si alguno de los campos del formulario está vacío aparecerá una pantalla informando que faltan campos. - En el caso de que el usuario ya exista el guardado no se llevará a cabo y se avisará mediante una pantalla emergente.
Postcondiciones	- Se guardará el usuario en la base de datos de la aplicación. - Se mostrará una ventana informando de que el guardado se llevó a cabo correctamente.

Tabla 11.Casos de uso, modificar usuario

Nombre Caso de Uso	**Modificar un usuario**
Descripción	Permite modificar la información de los usuarios existentes en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un usuario a modificar. - El usuario a modificar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en pantalla.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Usuarios" → "Modificar o Eliminar Usuarios" para acceder a la ventana de "Modificar o Eliminar Usuarios". Al seleccionar un usuario del combo box que aparece, se cargarán los datos del usuario seleccionado en los campos. Una vez modificados los campos se pulsará el botón "Guardar Modificación".
Excepciones	Si alguno de los campos del formulario se modifica a vacío aparecerá una pantalla informando que faltan campos en el formulario.
Postcondiciones	- Se guardará las modificaciones del usuario en la base de datos de la aplicación. - Se mostrará una ventana informando de que la modificación se llevó a cabo correctamente.

Tabla 12.Casos de uso, eliminar usuario

Nombre Caso de Uso	**Eliminar un usuario**
Descripción	Permite marcar como eliminado un usuario existente en la base de datos.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un usuario a eliminar. - El usuario que se desea eliminar debe existir en la base de datos de la aplicación y no estar marcado como eliminado. - Su información debe estar cargada en la pantalla.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Usuarios" → "Modificar o Eliminar Usuarios" para acceder a la ventana de "Modificar o Eliminar Usuarios". Al seleccionar un usuario del combo box que aparece, se cargarán los datos del usuario seleccionado en los campos. Una vez modificados los campos se pulsará el botón "Eliminar Usuario".
Excepciones	No se contemplan
Postcondiciones	- Se guardará el estado de "eliminado" del usuario en la base de datos de la aplicación. - Se limpiarán los campos del formulario y se actualizará la lista de usuarios que se muestra en el combo box. - se mostrará una ventana informando de que la eliminación se llevó a cabo correctamente.

Tabla 13.Casos de uso, listado usuarios

Nombre Caso de Uso	**Listado de usuarios**
Descripción	Muestra una lista de los usuarios existentes en la aplicación, solo mostrará los usuarios que no estén marcados como eliminados.
Actores	Administrador.
Precondiciones	No se contemplan
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Usuarios" → "Listado de Usuarios" para acceder a la pantalla de "Listado de Usuarios".
Excepciones	En caso de no existir ningún usuario la tabla se mostrará vacía.
Postcondiciones	No se contemplan

4.3.1.4. Casos de uso de préstamos

El siguiente diagrama muestra las acciones que puede realizar el administrador sobre los préstamos:

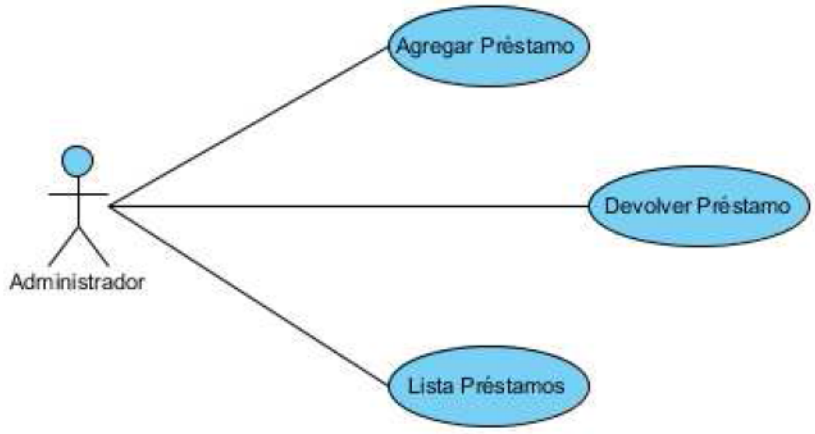


Ilustración 9. Casos de uso préstamos

A continuación detallaremos correctamente cada uno de los casos de uso incluidos dentro del diagrama.

Tabla 14.Casos de uso, agregar préstamo

Nombre Caso de Uso	**Agregar un nuevo préstamo**
Descripción	Crear un nuevo préstamo en la base de datos para notificar que a un usuario se le prestó un libro.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un usuario y un libro. - El Usuario y el libro seleccionados deben existir en la base de datos de la aplicación y no estar marcado como eliminado. - EL libro debe estar disponible (disponer de copias del mismo). A su vez el usuario no puede tener 3 préstamos a la hora de realizar uno nuevo.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Préstamos" → "Agregar Préstamo" para acceder a la ventana de "Agregar Préstamo Nuevo". Rellenará todo el formulario y le dará al botón de "guardar".
Excepciones	- Si alguno de los campos del formulario está vacío aparecerá una pantalla informando que faltan campos.
Postcondiciones	- Se guardará el préstamo en la base de datos de la aplicación. - Se mostrará una ventana informando de que el guardado se llevó a cabo correctamente.

Tabla 15.Casos de uso, devolver préstamo

Nombre Caso de Uso	**devolver un préstamo**
Descripción	Permite marcar como "devuelto" un préstamo existente en la base de datos. Las copias del libro aumentarán en 1 y los libros prestados del usuario disminuirán en 1 unidad.
Actores	Administrador.
Precondiciones	- Es necesario seleccionar un usuario. - Al seleccionar un usuario se cargarán los libros que tiene prestados y se seleccionará uno de los libros que le fueron prestados al usuario. - Es necesario que el usuario al menos tenga 1 libro prestado.
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú "Préstamos" → "Devolver Préstamo" para acceder a la ventana de "Devolver Préstamos". Seleccionará un usuario y un libro y se le dará al botón de "Devolver Préstamo".
Excepciones	No se contemplan.
Postcondiciones	- Se marcará el préstamo como "devuelto" en la base de datos de la aplicación. - Se mostrará una ventana informando de que la devolución se llevó a cabo correctamente.

Tabla 16.Casos de uso, listado préstamo

Nombre Caso de Uso	**Listado de préstamos**
Descripción	Muestra una lista de los préstamos existentes en la aplicación, solo mostrará los préstamos que no estén marcados como “devueltos”.
Actores	Administrador.
Precondiciones	No se contemplan
Flujo de acciones	En la ventana inicial desde el menú superior el administrador dará al menú “Préstamos” → “Listado de Préstamos” para acceder a la pantalla de “Listado de Préstamos”.
Excepciones	En caso de no existir ningún préstamo la tabla se mostrará vacía.
Postcondiciones	No se contemplan

4.3.2. Diagrama de clases de alto nivel

En este apartado se incluirá un diagrama de clases de alto nivel, en el cual podremos apreciar una idea general de cómo se distribuirán las clases Java que compondrán la aplicación. Para esta primera organización de clase las agruparemos por los principales paquetes de los que estará formada la aplicación, siendo posible agregar nuevos paquetes o sub-paquetes a los que indicamos, de la misma forma que con los paquetes, será necesario incluir más clases de las indicadas. EL siguiente diagrama muestra una idea general de la organización de la aplicación:

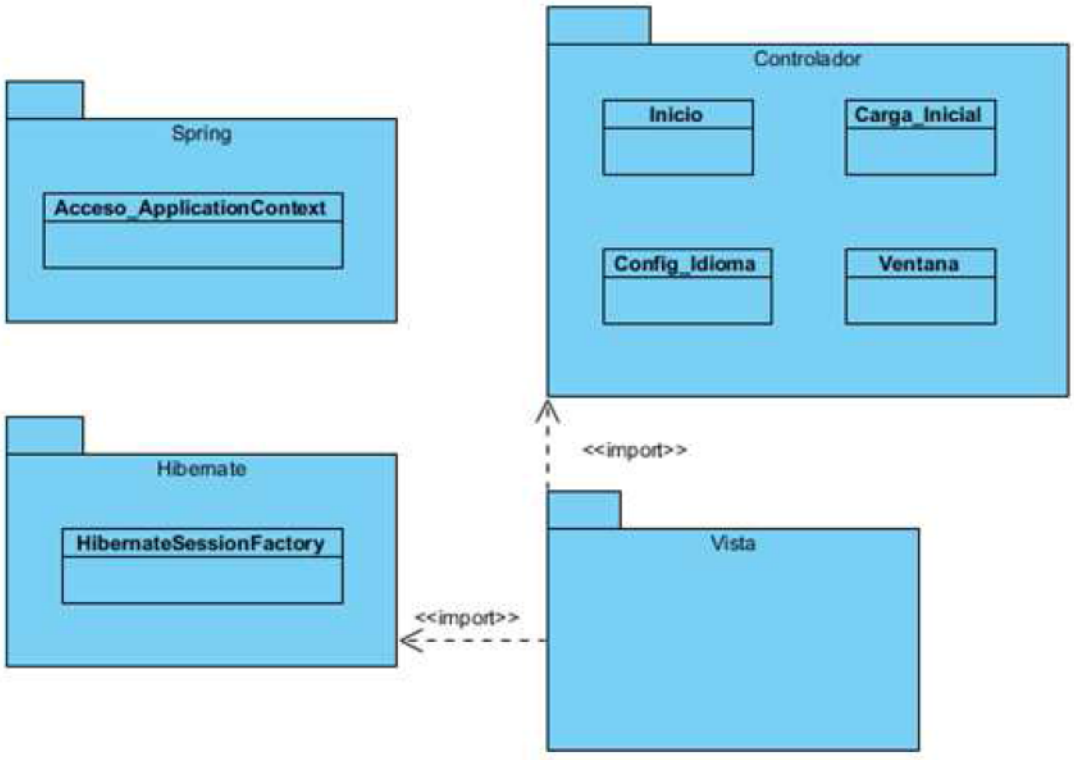


Ilustración 10. Diagrama clases alto nivel

Como se decidió usar el modelo MVC (Modelo, Vista, Controlador), la distribución de los principales paquetes y clases de la aplicación intentará en la medida de lo posible ser coherente con el mismo. Por este motivo los principales paquetes serán los siguientes:

- Modelo: Corresponde a los paquetes Controlador e Hibernate del diagrama. El paquete Hibernate es el encargado de realizar la gestión con la base de datos y el paquete Controlador es el encargado de iniciar la aplicación, cargar la ventana principal y cargar el idioma de la aplicación.

- Vista: Corresponde al paquete Vista en el anterior diagrama.

- Controlador: que corresponde al paquete Spring en el diagrama.

A continuación trataremos con un poco más de profundidad los paquetes definidos anteriormente, aunque es muy posible que a lo largo de la aplicación se creen nuevos paquetes y clases adicionales a estos.

4.3.2.1. Controlador

El paquete Controlador es el punto de partida de la aplicación. Nuestra clase Inicio será la que contenga en método "main" que será el encargado de iniciar el programa. En esta misma clase Inicio definiremos el lugar donde se encuentra el fichero "applicationContext.xml" necesario para el correcto funcionamiento del framework Spring.

La clase Inicio llamará a la clase Carga_Inicial y esta será la encargada de solicitar a Spring la creación de los objetos Config_Idioma y Ventana. El objeto Config_Idioma será el encargado de cargar el idioma por defecto en la aplicación, mientras que Ventana se encargará de crear la ventana principal de la aplicación.

4.3.2.2. Hibernate

Dentro del paquete Hibernate tendremos la clase de configuración de sesiones de Hibernate "HibernateSessionFactory" así como del fichero de configuración de Hibernate que le llamaremos "hibernate.cfg.xml".

Otra de las cosas a incluir en este paquete son todas las clases de persistencia que serán capaces de relacionar los objetos Java con la información existente o información que se quiera modificar/eliminar de las tablas de la Base de Datos. Los objetos mencionados anteriormente también se incluirán en este paquete.

4.3.2.3. Vista

En el diagrama el paquete Vista aparece vacío, pero en la aplicación final no será así. En este paquete se dispondrán de todas las ventanas necesarias para la correcta utilización de la aplicación.

Debido al gran número de ventanas, clases, interfaces y ficheros idiomáticos a realizar en el paquete de vista es muy recomendable ordenar las clases por temáticas de la siguiente forma:

- Sub-paquete Configuración: contendrá todo lo necesario para crear la ventana de configuración de idiomas de la aplicación.
- Sub-paquete escritores: contendrá todo lo necesarias para crear la ventana o ventanas que gestionarán los escritores de la aplicación.
- Sub-paquete libros: contendrá todo lo necesarias para crear la ventana o ventanas que gestionarán los libros de la aplicación.
- Sub-paquete préstamos: contendrá todo lo necesarias para crear la ventana o ventanas que gestionarán los préstamos de la aplicación.
- Sub-paquete usuarios: contendrá todo lo necesarias para crear la ventana o ventanas que gestionarán los usuarios de la aplicación.

De esta forma la aplicación tendrá una correcta ordenación y será más sencillo realizar futuras acciones de mantenimiento de ventanas sobre ella.

4.3.2.4. Spring

Este paquete será el encargado de la creación de los objetos en la aplicación, así como la gestión de las transacciones y la resolución de dependencias de clases.

Se dispondrá dentro de este paquete del fichero de configuración de Spring al cual le llamaremos “applicationContext.xml”.

Otra de las cosas que introduciremos en este paquete es la clase auxiliar “Acceso_ApplicationContext”, cuya finalidad es la de poder acceder al contexto de Spring siempre que se quiera y así poder crear los objetos de la aplicación mediante el framework Spring.

4.3.3. Modelo entidad – relación

En esta primera fase análisis, el modelo entidad-relación representa la organización lógica de los datos presentes en la Base de datos de la aplicación, no se entrará en detalle en la forma en la que se tratarán los datos ni de qué tipo serán los mismo.

En el siguiente diagrama se mostrará el modelo entidad-relación con las principales tablas que usará la aplicación:

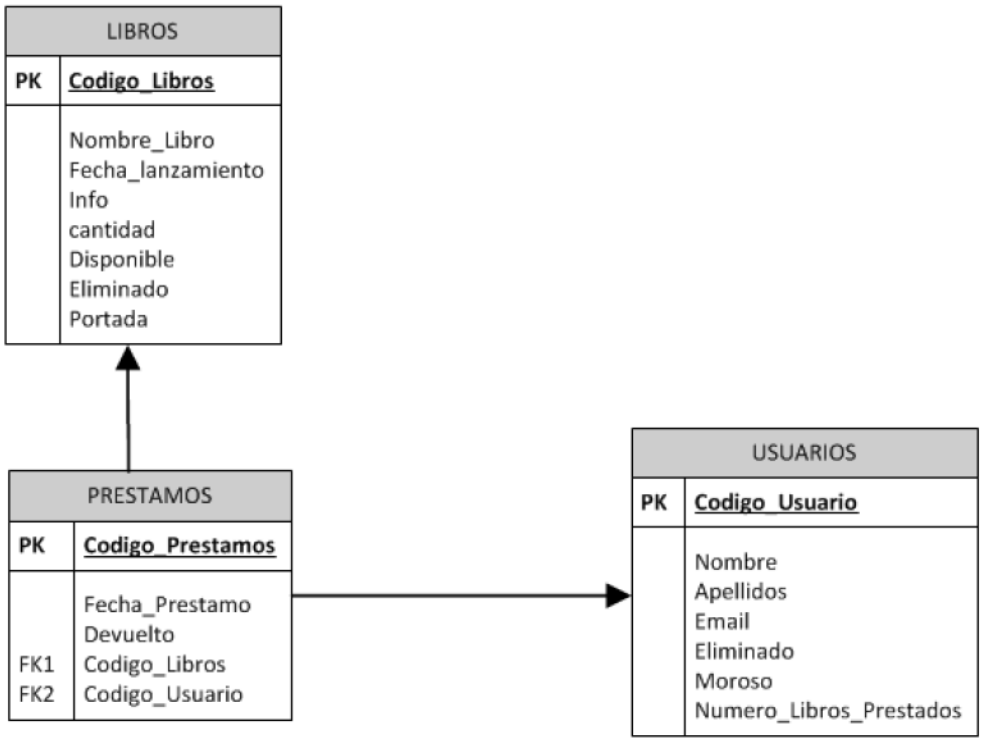


Ilustración 11. Modelo entidad - relación

Estas 3 entidades son las tablas principales en las que se basará en funcionamiento de la aplicación. Estas tablas no serán las únicas presentes, se necesitarán al menos entidades para escritores, géneros, idiomas y editoriales para completar diferentes datos referentes a libros. Estas tablas se definirán más adelante.

En el diagrama se aprecia que existen 3 entidades y 2 relaciones entre las mismas. De esta forma se tiene una tabla Usuarios y una tabla Libros y la tabla Préstamos hace de relación entre las anteriores para asignar un libro a un usuario, la tabla préstamos contendrá datos sobre el préstamo a realizar aparte de las clases de usuario y libro.

A continuación explicaremos más a fondo las 3 entidades que se muestran en el diagrama" ilustración 12. Modelo entidad - relación"

4.3.3.1. Usuarios

Esta entidad representa los usuarios del servicio los cuales podrán realizar préstamos de libros existentes. Esta entidad contiene toda la información personal necesaria para identificar a cada uno de los usuarios. Los atributos que presenta son los siguientes:

- Codigo_Usuario: Será el identificador único de cada usuario. Este dato no podrá modificarse, se agregará automáticamente por la aplicación.
- Nombre: representa el nombre del usuario.
- Apellidos: atributo que representa el apellido del usuario correspondiente.
- Email: dirección de correo del usuario.
- Eliminado: muestra si el usuario está activo o no en la aplicación.
- Moroso: atributo que representa si el usuario se excedió en fecha en la devolución de un préstamo.
- Numero_Libros_Prestados: número de libros que tiene en propiedad el usuario en ese momento concreto.

4.3.3.2. Libros

Identifica los diferentes libros que se disponen en la aplicación para que los usuarios hagan uso de los mismos. Esta tabla contiene más atributos de los presentes en la ilustración 11, pero por simplificar el diagrama no se incluyeron. En la siguiente lista de atributos si incluiremos dichos datos que omitimos y se dará una breve explicación de cada uno de ellos. Los atributos presentes son los siguientes:

- Codigo_Libros: será el identificador único de cada libro. Este dato no podrá modificarse, será agregado automáticamente por la aplicación.

- Nombre_Libro: representa el título del libro.
- Fecha_Lanzamiento: fecha en la que se publicó el libro en cuestión.
- Info: Información o sinopsis del libro.
- Cantidad: representa la cantidad de ejemplares que se disponen del libro.
- Disponible: atributo que representa que el libro puede ser prestado pero que en ese mismo momento no hay copias disponibles del mismo.
- Eliminado: atributo que identifica si un libro ya no está disponible para ser prestado o está descatalogado.
- Portada: Atributo que almacena imagen de portada del libro.

Hasta aquí hemos definido los atributos presentes en el diagrama, a continuación presentaremos los atributos que no están presentes y están relacionados con otras entidades que no están presentes en el diagrama y de las cuales se hablará más adelante:

- Codigo_Genero_Li: relaciona un libro con uno de los géneros presentes en la Base de Datos.
- Codigo_Edit_Li: relaciona un libro con una de las editoriales presentes en la Base de Datos.
- Codigo_Escritor_Li: relaciona un libro con uno de los escritores presentes en la Base de Datos.
- Codigo_Idioma_Li: relaciona un libro con uno de los idiomas presentes en la Base de Datos.

Todos estos atributos serán claves foráneas (FK) de la entidad

4.3.3.3. Préstamos

Esta entidad representa el préstamo de un libro por el centro a uno de los usuarios existentes en la aplicación. Los atributos que posee esta entidad son los siguientes:

- Codigo_Prestamos: será el identificador único de cada uno de los préstamos. Este dato no podrá modificarse, será agregado automáticamente por la aplicación.
- Fecha_Prestamo: Atributo que identifica la fecha en la que se realiza el préstamo.
- Devuelto: Identifica si el préstamo fue devuelto o no.
- Codigo_Libros: relaciona el préstamo con el libro que va a ser prestado. El libro tiene que estar presente en la base de datos y tiene que estar disponible dicho libro.
- Codigo_usuario: relaciona el préstamo con el usuario que realizará el mismo. El usuario no puede tener más de 3 préstamos a la vez por lo que en el momento del préstamo en el atributo “Numero_Libros_Prestados” de la entidad “usuarios” no puede tener el valor de 3 o superior.

4.4. Diseño

A continuación se procederá a la explicación lo más detallado posible tanto del código de la aplicación como de la base de datos creada para la misma.

En el apartado de análisis se pretendía una idea general de la estructura de la aplicación para tener unas pautas desde las cuales crear la aplicación. El diseño de la aplicación es escrito después de crear la aplicación y en él se pretende analizar y documentar a fondo las clases y entidades creadas en la creación de la aplicación.

4.4.1. Modelo de datos

En la parte de análisis se estableció un modelo Entidad-Relación de alto nivel, de esa forma se consiguió dar una ligera idea de cómo estaría distribuida y las relaciones principales de nuestra Base de Datos.

En este nuevo apartado estudiaremos en su totalidad el modelo de datos, indicando todas las entidades y relaciones que se crearán, todos los atributos disponibles, el tipo de cada atributo y las diferentes claves primarias (PK) y claves foráneas (FK) de cada entidad.

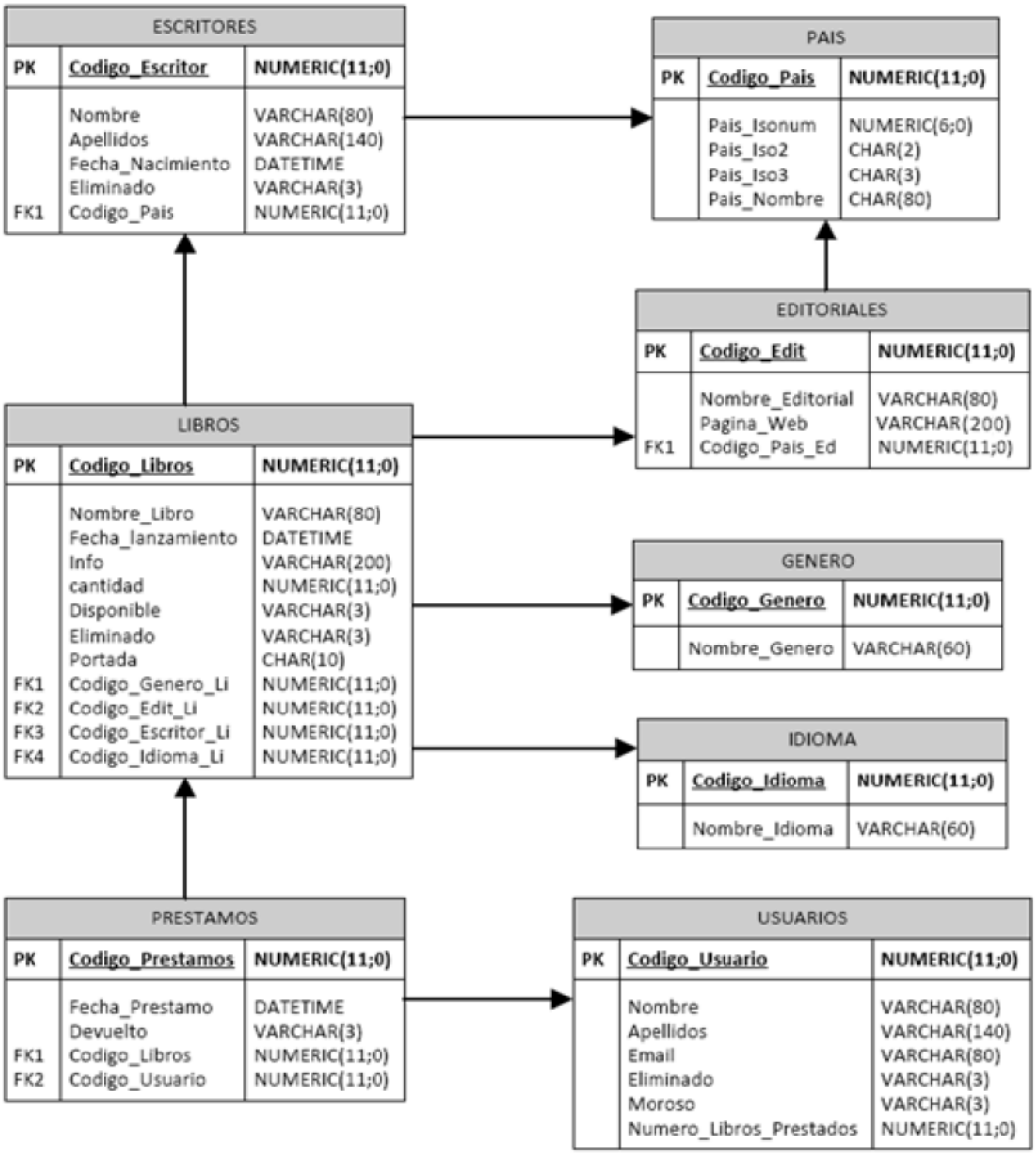


Ilustración 13. Modelo de datos

Como se puede apreciar en la figura “Ilustración 14. Modelo de datos” el modelo está compuesto por 8 entidades y 8 relaciones. Antes de explicar cada entidad por separado diremos que en Oracle no hay tipo de datos autoincrementados, sino que existen unos objetos llamados “sequence” que hace el mismo trabajo, con la diferencia de que se crear por separado de las entidades. Por lo que se creará un “sequence” para cada tabla y se le asignará a cada una de las entidades para que sus claves primarias sean únicas.

La gran mayoría de las entidades nuevas sirven para dar más funcionalidades e información a la entidad Libro, de esta forma las entidades Escritores, Editoriales, Genero e Idioma servirán para dar más información a los objetos libro presentes en la entidad Libro De la misma forma la entidad País servirá para agregar nueva información a Escritores y Editoriales.

En el Anexo ANEXO6 se dispondrá de todo el código SQL para la creación de todas las tablas, relaciones y sequence que se exponen en este apartado. A continuación explicaremos fondo las 8 entidades que se muestran con sus respectivas relaciones:

4.4.1.1. Usuarios

USUARIOS		
PK	**Codigo_Usuario**	**NUMERIC(11;0)**
	Nombre	VARCHAR(80)
	Apellidos	VARCHAR(140)
	Email	VARCHAR(80)
	Eliminado	VARCHAR(3)
	Moroso	VARCHAR(3)
	Numero_Libros_Prestados	NUMERIC(11;0)

Ilustración 15. Entidad usuarios

Esta entidad representa los usuarios del servicio los cuales podrán realizar préstamos de libros existentes. Esta entidad contiene toda la información personal necesaria para identificar a cada uno de los usuarios. Los atributos que presenta son los siguientes:

- Codigo_Usuario: Será la clave primaria o PK, también se le puede definir como el identificador único de cada usuario. Este atributo nunca será nulo y se le asignará un sequence tal y como se explicó anteriormente. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad, este dato nunca será modificable. El tipo de dato de este atributo será "NUMERIC(11;0)" pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre: representa el nombre del usuario. El tipo de dato del atributo nombre será "VARCHAR(80)" pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 80 caracteres.
- Apellidos: atributo que representa el apellido del usuario correspondiente. Su tipo de dato será "VARCHAR(140)".
- Email: Este atributo será del tipo "VARCHAR(80)" y contendrá la dirección de correo electrónico del usuario.
- Eliminado: muestra si el usuario está activo o no en la aplicación. Su tipo de dato será "VARCHAR(3)" y únicamente podrá contener las cadenas de caracteres "SI" o "NO", de esta forma se identificará si el usuario se encuentra eliminado (SI) o está activo (NO).

- Moroso: atributo que representa si el usuario se excedió en fecha en la devolución de un préstamo o tiene libros en propiedad fuera de fecha. De igual forma que en el atributo “Eliminado” el atributo moroso será del tipo “VARCHAR(3)” conteniendo la cadena de caracteres “SI” si el usuario es moroso o “NO” para un usuario normal.
- Numero_Libros_Prestados: número de libros que tiene en propiedad el usuario en ese momento concreto. El tipo de dato de este atributo será “NUMERIC(11;0)”.

4.4.1.2. Libros

LIBROS		
PK	**Codigo_Libros**	**NUMERIC(11;0)**
	Nombre_Libro	VARCHAR(80)
	Fecha_lanzamiento	DATETIME
	Info	VARCHAR(200)
	cantidad	NUMERIC(11;0)
	Disponible	VARCHAR(3)
	Eliminado	VARCHAR(3)
	Portada	CHAR(10)
FK1	Codigo_Genero_Li	NUMERIC(11;0)
FK2	Codigo_Edit_Li	NUMERIC(11;0)
FK3	Codigo_Escritor_Li	NUMERIC(11;0)
FK4	Codigo_Idioma_Li	NUMERIC(11;0)

Ilustración 16. Entidad libros

Identifica los diferentes libros que se disponen en la aplicación para que los usuarios hagan uso de los mismos. Esta tabla contiene más atributos de los presentes en el diagrama 5.5., pero por simplificar el diagrama no se incluyeron. En la siguiente lista de atributos si incluiremos dichos datos que omitimos y se dará una breve explicación de cada uno de ellos. Los atributos presentes son los siguientes:

- Codigo_Libros: Será la clave primaria, también se le puede definir como el identificador único de cada libro. Este atributo nunca será nulo y se le asignará un sequence tal y como se dijo anteriormente. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad, este dato nunca será modificable. El tipo de dato de este atributo será "NUMERIC(11;0)" pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre_Libro: representa el título del libro. El tipo de dato de este atributo será "VARCHAR(80)" pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 80 caracteres.
- Fecha_Lanzamiento: fecha en la que se publicó el libro. El tipo de dato que se quiere contener es una fecha por lo que el de dato de este atributo será "DATETIME".

- Info: contendrá la información o sinopsis del libro, el tipo de dato será "VARCHAR(200)".
- Cantidad: Este atributo tendrá un tipo de dato "NUMERIC(11;0)" y representa la cantidad de ejemplares que se disponen del libro.
- Eliminado: muestra si el libro está activo o no en la aplicación. Su tipo de dato será "VARCHAR(3)" y únicamente podrá contener las cadenas de caracteres "SI" o "NO", de esta forma se identificará si el usuario se encuentra eliminado (SI) o está activo (NO).
- Disponible: de la misma forma que el atributo anterior este atributo será del tipo "VARCHAR(3)". Contendrá las cadenas de caracteres "SI" o "NO", "SI" en caso de que existan ejemplares disponibles del libro, "NO" si el atributo "Cantidad" tiene un valor de "0".
- Portada: Atributo que almacena imagen de portada del libro. En este caso se usará un tipo de dato "BLOB" en cual puede contener datos binarios que se asignan a una matriz de tipo Byte, de esta forma se podrá almacenar las portadas de los libros.
- Codigo_Genero_Li: Es una clave foránea (FK) con la que se representa la relación entre un libro y el género principal del mismo. Este atributo contendrá el valor de una PK de la entidad Genero, por lo que el tipo de dato de este atributo tiene que ser obligatoriamente igual que la PK de la entidad Genero, "NUMERIC(11;0)". El valor presente en este campo tiene que ser un valor existente de PK de la entidad Genero.
- Codigo_Edit_Li: Al igual que el atributo anterior, este atributo también corresponde a una FK, representa la relación entre un libro y la editorial al que le pertenece el libro. Del mismo modo que antes, el valor presente será "NUMERIC(11;0)" y obligatoriamente tendrá que existir el valor como PK de la entidad Editoriales.
- Codigo_Escritor_Li: Una nueva clave foránea, es representará la relación entre las entidades Libros y Escritores, indicando que libros o libros fueron escritos por un autor. Como en los dos casos anteriores este atributo también será del

"NUMERIC(11;0)" al igual que la clave primaria de la tabla Escritores. El valor presente tendrá que existir como dato en una PK de la tabla Escritores.

- Codigo_Idioma_Li: Es el último atributo de nuestra entidad Libros, corresponde a una FK y representa la relación entre un libro y el idioma en que está disponible el mismo en nuestra biblioteca. Como todas nuestras FK será del tipo "NUMERIC(11;0)" para coincidir con el tipo de datos de la PK de la entidad con la que está relacionada, en este caso la entidad es Idioma. Al igual que en todas las FK anteriores el dato presente en esta tendrá que existir como PK en la entidad Idioma de la Base de Datos para que todo funcione correctamente.

4.4.1.3. Préstamos

PRESTAMOS		
PK	**Codigo_Prestamos**	**NUMERIC(11;0)**
	Fecha_Prestamo	DATETIME
	Devuelto	VARCHAR(3)
FK1	Codigo_Libros	NUMERIC(11;0)
FK2	Codigo_Usuario	NUMERIC(11;0)

Ilustración 17. Entidad préstamos

Esta entidad representa el préstamo de un libro por el centro a uno de los usuarios existentes en la aplicación. La tabla contendrá 2 claves foráneas (FK) que relacionarán el préstamo con un usuario y un libro. Los atributos que posee esta entidad son los siguientes:

- Codigo_Prestamos: Será la clave primaria o PK, también se le puede definir como el identificador único de cada préstamo. Este atributo nunca será nulo, no será modificable y se le asignará un sequence tal y como se explicó anteriormente. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad. El tipo de dato de este atributo será "NUMERIC(11;0)" pudiendo de esta forma tener números de 0 a 11 dígitos.
- Fecha_Prestamo: Atributo que identifica la fecha en la que se realiza el préstamo. Al querer contener una fecha el tipo de dato de este atributo será "DATETIME".
- Devuelto: Identifica de si el préstamo fue devuelto o no. Su tipo de dato será "VARCHAR(3)" y únicamente podrá contener las cadenas de caracteres "SI" o "NO", de esta forma se identificará si el usuario se encuentra eliminado (SI) o está activo (NO).
- Codigo_Libros: Es una clave foránea (FK) de la tabla libros y representa la relación entre un libro y un préstamo. Este atributo contendrá el valor de una PK de la tabla libros, por lo que el tipo de dato de este atributo tiene que ser obligatoriamente igual que la PK de la entidad Libros, "NUMERIC(11;0)". El

valor presente en este campo tiene que ser un valor existente de PK de la entidad Libros.

- Codigo_usuario: Al igual que “Codigo_Libros” este atributo es una FK y en este caso representará la relación entre el préstamo y el usuario que realiza el préstamo. El tipo de dato será “NUMERIC(11;0)” de tal forma que coincida con el tipo de dato de la PK de la entidad Usuarios. Del mismo modo que con el anterior atributo, el dato presente tiene que existir como PK en la tabla de Usuarios.

4.4.1.4. Escritores

ESCRITORES		
PK	**Codigo_Escritor**	**NUMERIC(11;0)**
	Nombre	VARCHAR(80)
	Apellidos	VARCHAR(140)
	Fecha_Nacimiento	DATETIME
	Eliminado	VARCHAR(3)
FK1	Codigo_Pais	NUMERIC(11;0)

Ilustración 18. Entidad escritores

Esta entidad representa los escritores, los cuales podrán tener asignados de 0 a N libros pero un libro tan solo podrá tener un escritor asignado por lo que la relación entre las entidades Escritores y Libros será 1:N.

Esta tabla contendrá una FK que mostrará una relación entre un escritor y la entidad país y representará le país de nacimiento del escritor-

El usuario final de la aplicación será libre de crear escritores nuevos o modificar los escritores existentes en la Base de Datos. Los atributos que posee esta entidad son los siguientes:

- Codigo_Escritor: Será la clave primaria o PK. Este atributo nunca será nulo y se le asignará un sequence. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad. Este dato nunca será modificable por el administrador de la base de datos. El tipo de dato de este atributo será “NUMERIC(11;0)” pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre: representa el nombre del autor. El tipo de dato del atributo nombre será “VARCHAR(80)” pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 80 caracteres.
- Apellidos: atributo que representa el apellido o apellidos del autor. El tipo de dato del atributo nombre será “VARCHAR(140)” pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 80 caracteres.

- Fecha_Nacimiento: Atributo que identifica la fecha de nacimiento del autor. El tipo de dato que se quiere contener es una fecha por lo que el de dato de este atributo será “DATETIME”.
- Eliminado: muestra si el usuario está activo o no en la aplicación. Su tipo de dato será “VARCHAR(3)” y únicamente podrá contener las cadenas de caracteres “SI” o “NO”, de esta forma se identificará si el usuario se encuentra eliminado (SI) o está activo (NO).
- Codigo_Pais: Es una clave foránea (FK) con la cual se representa la relación entre el escritor y el país de nacimiento de este. Este atributo contendrá el valor de una PK de la entidad País, por lo que el tipo de dato de este atributo tiene que ser obligatoriamente igual que la PK de la entidad País, “NUMERIC(11;0)”. El valor presente en este campo tiene que ser un valor existente de PK de la entidad País.

4.4.1.5. Editoriales

EDITORIALES		
PK	**Codigo_Edit**	**NUMERIC(11;0)**
	Nombre_Editorial	VARCHAR(80)
	Pagina_Web	VARCHAR(200)
FK1	Codigo_Pais_Ed	NUMERIC(11;0)

Ilustración 19. Entidad editoriales

Esta entidad representa la editorial la cual publicó un libro en cuestión. Esta relación será 1:N de tal forma que un libro solo puede tener una editorial, pero una editorial puede tener varios libros.

Esta entidad no será modificable por el usuario final de la aplicación, sino que tendrá una lista de editoriales preestablecida por el administrador de la base de datos. Los atributos que posee esta entidad son los siguientes:

- Codigo_Edit: Será la clave primaria o PK. Este atributo nunca será nulo y se le asignará un sequence. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad. Este dato nunca será modificable por el administrador de la base de datos. El tipo de dato de este atributo será "NUMERIC(11;0)" pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre_Editorial: representa el nombre de la editorial. El tipo de dato del atributo nombre será "VARCHAR(80)" pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 80 caracteres.
- Pagina_Web: representa la dirección de página web de la editorial. Este atributo será del tipo "VARCHAR(200).
- Codigo_Pais_Ed: Es una clave foránea (FK) con la cual se representa la relación entre una editorial y el país de la misma. Este atributo contendrá el valor de una PK de la entidad País, por lo que el tipo de dato de este atributo tiene que ser obligatoriamente igual que la PK de la entidad País, "NUMERIC(11;0)". El

valor presente en este campo tiene que ser un valor existente de PK de la entidad País.

4.4.1.6. Genero

GENERO		
PK	Codigo_Genero	NUMERIC(11;0)
	Nombre_Genero	VARCHAR(60)

Ilustración 20. Entidad género

Esta entidad representa el género que puede tener cada libro. Inicialmente esta relación es 1:1 lo que quiere decir que un libro solo puede tener asignado un género, en futuras mejoras se podría plantear que esta relación fuese 1:N de forma que un libro pudiese tener un género principal y/o varios géneros secundarios en cada libro presente en la Base de Datos.

Esta entidad no será modificable por el usuario final de la aplicación, sino que tendrá una lista de géneros preestablecida por el administrador de la base de datos. Los atributos que posee esta entidad son los siguientes:

- Codigo_Genero: Será la clave primaria o PK, también se le puede definir como el identificador único de cada género. Este atributo nunca será nulo y se le asignará un sequence. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad, este dato nunca será modificable. El tipo de dato de este atributo será "NUMERIC(11;0)" pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre_Genero: representa el nombre del género. El tipo de dato del atributo nombre será "VARCHAR(60)" pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 60 caracteres.

4.4.1.7. País

PAIS		
PK	**Codigo_Pais**	**NUMERIC(11;0)**
	Pais_Isonum	NUMERIC(6;0)
	Pais_Iso2	CHAR(2)
	Pais_Iso3	CHAR(3)
	Pais_Nombre	CHAR(80)

Ilustración 21. Entidad país

Esta entidad representa el país de procedencia de los escritores o el país al que pertenece una editorial. Las relaciones con esta tabla serán N:1, de tal forma que 1 escritor/editorial solo podrá tener un país pero 1 país podrá pertenecer a más de un escritor/editorial. No se contemplan dobles nacionalidades ni excepciones similares.

Esta entidad no será modificable por el usuario final de la aplicación, sino que tendrá una lista de géneros preestablecida por el administrador de la base de datos.

Esta entidad tiene una curiosa distribución de atributos y es debido a que gracias a un antiguo profesor nos facilitó un listado completo de todos del mundo con todos los datos necesarios para nuestra aplicación. Los datos que se encuentran en esta entidad están completamente estandarizados y contiene los 240 países que existen en la actualidad con sus códigos internacionales y su nombre.

Los atributos que posee esta entidad son los siguientes:

- Codigo_Pais: es la clave primaria. Al tener todos los insert ya disponibles para la tabla no es necesario la creación de un sequence para este atributo. El tipo de dato de este atributo será “NUMERIC(11;0)” pudiendo de esta forma tener números de 0 a 11 dígitos.
- Pais_Isosum: Este atributo representa el código numérico internacional que posee cada país. El tipo de dato de este atributo será “NUMERIC(6;0)” pudiendo de esta forma tener números de 0 a 6 dígitos.
- Pais_Iso2: representa el código alfanumérico de 2 caracteres con el cual se identifica al país en cuestión. El tipo de dato usado es “VARCHAR(2)”

- Pais_Iso3: De la misma forma que el anterior atributo, este atributo representa el código alfanumérico que representará al país pero esta vez utilizando 3 caracteres en vez de 2. El tipo de dato usado es “VARCHAR(3)”
- Pais_Nombre: representa el nombre del país. El tipo de dato del atributo nombre será “VARCHAR(60)” pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 60 caracteres.

4.4.1.8. Idioma

IDIOMA		
PK	**Codigo_Idioma**	**NUMERIC(11;0)**
	Nombre_Idioma	VARCHAR(60)

Ilustración 22. Entidad idioma

Esta entidad representa el idioma que puede tener cada libro. Inicialmente esta relación es 1:N lo que quiere decir que un libro solo puede tener un idioma, pero un idioma puede pertenecer a varios libros En futuras mejoras se podría plantear que esta relación fuese N:N de forma que un libro pudiese tener 1 o varios idiomas.

Esta entidad no será modificable por el usuario final de la aplicación, sino que tendrá una lista de idiomas preestablecida por el administrador de la base de datos. Los atributos que posee esta entidad son los siguientes:

- Codigo_Idioma: Será la clave primaria o PK, también se le puede definir como el identificador único de cada idioma. Este atributo nunca será nulo y se le asignará un sequence. De esta forma el valor del dato será asignado por el sequence creado y siempre será un valor único dentro de la entidad, este dato nunca será modificable. El tipo de dato de este atributo será “NUMERIC(11;0)” pudiendo de esta forma tener números de 0 a 11 dígitos.
- Nombre_Idioma: representa el nombre del idioma. El tipo de dato del atributo nombre será “VARCHAR(60)” pudiendo incluir cualquier tipo de carácter alfanumérico hasta un máximo de 60 caracteres.

4.4.2. Estructura y contenido de clases

La aplicación se compone de 51 clases Java, 23 interfaces, 9 ficheros XML de configuración y 56 archivos properties para la configuración de la idiomatización de la aplicación. Todos ellos están repartidos en 12 paquetes o sub-paquetes.

Realizaremos las diferentes explicaciones del código siguiendo la estructura de los paquetes creados. A continuación procederemos a explicar los componentes de la aplicación.

4.4.2.1. Paquete Controlador

Inicio.java

Es la clase inicial de la aplicación, contiene en método "main()" que es llamado por la máquina virtual de java para comenzar el programa.

En esta clase evitaremos introducir cualquier tipo de código innecesario por lo que tan solo incluiremos en ella la ruta del fichero aplicationContext.xml para la configuración de Spring. Seguidamente llamaremos al método opciones_Defecto() de la clase carga_Inicial.java que se encarga de inicializar todos los parámetros necesarios, carga el idioma por defecto y llamar a la ventana de la aplicación.

Carga_Inicial.java

Esta clase se encargará de cargar las diferentes opciones por defecto de la aplicación.

Inicialmente llamará a un método propio de la clase que se llama establecer_Nimbus() en el que se carga un aspecto propio de Java Swing que hará más atractiva la interfaz gráfica de la aplicación.

Mediante Spring llamamos a la clase “vista_c.Idioma_por_defecto();” que como su nombre indica carga el idioma de la última vez que se ejecutó la aplicación, este método se explicará detenidamente más adelante.

Finalmente creamos una instancia de la clase Ventana.java que se encargará de crear los paneles y botones necesarios para el funcionamiento de la aplicación.

Config_Idioma.java

Esta clase implementa la interfaz IConfig_Idioma, la cual le obliga a incluir diferentes métodos en la misma. La interfaz es indispensable para hacer llamadas entre clases mediante Spring.

La clase contiene una variable de tipo privada de tipo String llamada “IDIOMA” que contiene en todo momento el idioma actual de la aplicación. La clase está compuesta por diferentes métodos, los cuales iremos enumerando y comentando a continuación:

- public void Idioma_por_defecto(): La aplicación contiene un fichero de persistencia llamado idioma.txt, en el cual se guarda el idioma seleccionado la última vez que se ejecutó la aplicación. Este método abre ese fichero .txt, recupera el dato que contiene y lo introduce en la variable IDIOMA.
- public void cambiar_Idioma_defecto(): Este método abre el fichero de texto mencionado anteriormente e introduce el valor actual de la variable IDIOMA en dicho fichero.
- public String getIDIOMA(): Método que devuelve el valor de la variable IDIOMA.
- public void cambio_Idioma_Aleman(),public void cambio_Idioma_ingles(),public void cambio_Idioma_español(): Los 3 métodos funcionan de la misma forma y se accionan cuando un usuario cambia el idioma seleccionado para la aplicación. Los métodos en cuestión cambian la variable IDIOMA dependiendo de cuál fue seleccionado y llaman al método “cambiar_Idioma_defecto()” para guardar en el fichero de persistencia el cambio de idioma de la aplicación.

IConfig_Idioma.java

Esta interfaz es implementada en la clase Config_Idioma.java la cual acabamos de comentar. Es completamente necesario el uso de interfaces para el uso de Spring en la aplicación.

En la interfaz tendremos que incluir obligatoriamente todos los métodos que queramos poder llamar fuera de la propia clase mediante Spring. Los métodos que son llamados dentro de la propia clase no es necesario incluirlos, por eso en nuestro caso el método "cambiar_Idioma_defecto()" no está incluido en la interfaz ya que no es llamado desde el exterior de la clase.

Por lo que nuestra interfaz la compondrían los siguientes métodos:

```
public void Idioma_por_defecto();
public String getIDIOMA();
public void cambio_Idioma_Aleman();
public void cambio_Idioma_ingles();
public void cambio_Idioma_español();
```

Ventana.java

Es la ventana principal de la aplicación. Contiene un menú superior completo y una imagen predefinida como portada.

El menú es una de las partes más importantes de la aplicación ya que desde él se realizaran las llamadas a las diferentes vistas que contiene la aplicación.

A continuación mostraremos una imagen de nuestro menú para poder explicar mejor el funcionamiento y creación del mismo

Ilustración 23. Menú Ventana

Primeramente se creará un JmenuBar "menuBar = new JMenuBar();" lo siguiente que se creará son los objetos JMenu que mediante el método add los agregaremos al menuBar. Lo último que se creará del menú serán los JMenuItem que también gracias al método add lo añadiremos al menú, pero esta vez los añadiremos al Jmenu

correspondiente no al JMenuBar, de esta forma se consigue una correcta ordenación en el menú. Creado ya nuestro menú solo nos falta pintarlo en la ventana usando en siguiente método "setJMenuBar(menuBar);"

En este estado los JMenuItem no realizan absolutamente nada por lo que les tendremos que agregar un escuchador para que su funcionamiento sea similar al de un botón normal, el escuchador que usaremos en este caso es el ActionListener (el cual tenemos que implementar dentro de la clase) y lo asignaremos a los diferentes ítems de la siguiente forma "prestamos_agregar.addActionListener(this);" siendo prestamos_agregar uno de nuestros JMenuItem.

Al hacer el "implements ActionListener" comentado anteriormente es obligatorio la introducción del método "public void actionPerformed(ActionEvent evento)" el cual será el encargado de gestionar los eventos que provienen de los JMenuItem o cualquier otro objeto de la aplicación. Mediante el atributo "evento" q recibe por parámetro el método podremos identificar el objeto que generó el evento y dependiendo del evento se podrán hacer diferentes acciones.

Otro aspecto importante de la clase ventana es el uso de "ResourceBundle" para la idiomatización de las ventanas. Todas las siguientes ventanas que se explique seguirán el mismo procedimiento, por ese motivo tan solo explicaremos dicho procedimiento esta vez. Antes de crear ninguna de los objetos de los que se compondrá la ventana de Swing comprobaremos el idioma por defecto, y dependiendo de este cargaremos un ResourceBundle diferente. La idea de usar este método es que se tiene diferentes archivos *.properties con los nombres de los diferentes objetos de los que está compuesta la ventana. Cada archivo properties estará en un idioma diferente de tal forma que dependiendo del archivo properties al que llamemos la ventana estará en un idioma o en otro. A continuación se muestra un pequeño ejemplo del código a utilizar.

```
 //CARGAMOS EL FICHERO PROPERTIES POR DEFECTO
private ResourceBundle rb= ResourceBundle.getBundle
("com.rivas.controlador.Ventana");

//CAMBIAMOS EL ResourceBundle AL PROPERTIES EN INGLES
rb = ResourceBundle.getBundle("com.rivas.controlador.Ventana_en");
```

```
//ASIGNAMOS EL VALOR DE Ventana.prestamos_devolver.text QUE ESTA DENTRO
DEL PROPERTIES AL JMENUITEM CREADO
prestamos_devolver = new JMenuItem
(rb.getString("Ventana.prestamos_devolver.text"));
```

De esta forma el valor de la clave del properties "Ventana.prestamos_devolver.text" será cargado en el JMenuItem y dependiendo de fichero cargado el texto del objeto cambiará.

4.4.2.2. Paquete hibernate

En este paquete incluiremos la clase que se encargará de servir de sesiones Hibernate a la aplicación, el fichero de configuración de Hibernate y todos los objetos Java junto a sus mapeados con los que Hibernate será capaz de utilizar los datos de la Base de Datos.

Debido a que todos los objetos Java y sus mapeos funcionan de la misma forma y tienen códigos extremadamente similares solo se comentará uno de ellos, de esta forma ahorramos tiempo al lector de leer varias veces lo mismo.

HibernateSessionFactory.java

Esta clase es una de las más importantes para el funcionamiento de Hibernate. Con ella serviremos a la aplicación de sesiones Hibernate que serán necesarias para realizar cualquier tipo de sentencia en la base de datos.

La clase tiene diversos atributos y métodos pero los más importantes son el atributo "CONFIG_FILE_LOCATION" al cual le tenemos que pasar la ruta exacta del fichero de configuración de hibernate del cual hablaremos más adelante.

En cuanto a los métodos de la clase tenemos el "getSession()" que devuelve una sesión de Hibernate, "rebuildSessionFactory()" que recarga la sesión ya creada y "closeSession()" que cierra la conexión.

hibernate.cfg.xml

Es el fichero de configuración de Hibernate y del cual ya hablamos en el apartado 3.1.1. En este fichero se le tiene que indicar el tipo de Base de datos que se usará, el tipo de conexión y el usuario y contraseña para conectarse a la base de datos.

Otro dato a incluir son las rutas de todos los ficheros descriptores de las entidades de la base de datos, los cuales indican a Hibernate la forma en la que se relacionarán los atributos de las entidades en la base de datos con los objetos Java.

Libros.java

Para la creación de esta clase tendremos que tener muy en cuenta la estructura de la entidad "libro" en la Base de Datos. Todos los datos tienen que ser lo más similares posible en ambos lugares. A continuación mostramos el resultado final del estudio de la entidad Libros y de la creación de los atributos en Libros.java.

LIBROS		
PK	**Codigo_Libros**	**NUMERIC(11;0)**
	Nombre_Libro	VARCHAR(80)
	Fecha_lanzamiento	DATETIME
	Info	VARCHAR(200)
	cantidad	NUMERIC(11;0)
	Disponible	VARCHAR(3)
	Eliminado	VARCHAR(3)
	Portada	CHAR(10)
FK1	Codigo_Genero_Li	NUMERIC(11;0)
FK2	Codigo_Edit_Li	NUMERIC(11;0)
FK3	Codigo_Escritor_Li	NUMERIC(11;0)
FK4	Codigo_Idioma_Li	NUMERIC(11;0)

```
private Long codigoLibros;
private String nombreLibro;
private Date fechaLanzamiento;
private String info;
private Long cantidad;
private String disponible;
private String eliminado;
private Blob portada;
private Genero genero;
private Editoriales editoriales;
private Escritores escritores;
private Idioma idioma;
```

Como podemos apreciar en todos los casos creamos tipos similares numeric/ Long, Varchar/ String, Datetime/ Date,…. En los único que variamos es en los casos en que el dato sea una clave foránea a otra clase, en tal caso crearemos un atributo del tipo del objeto que necesitamos tal y como están creados los 4 últimos atributos. Como nota diremos que antes de crear esta clase es necesario crear las clases con las cuales tiene dependencias.

Es necesario crear al menos un constructor para la clase, en este caso se incluirán 3 diferentes: un constructor completo con todos los atributos, otro mínimo con tan solo el codigoLibros y un constructor por defecto.

Por último incluiremos los métodos accesores (Get y Set) de todos los atributos presentes en la clase.

Libros.hbm.xml

Este será el fichero descriptor o de mapeo entre la clase Java Libros.java y la entidad de la Base de Datos Libros. Es completamente necesaria su creación para que Hibernate sepa enlazar los atributos de la base de datos con los de la clase.

Este tipo de ficheros ya fue comentado en el apartado 3.1.2 y en el ANEXO2 incluimos un ejemplo de su creación. Por ese motivo solo comentaremos las novedades que incluimos en este fichero.

El primero de las modificaciones incluidas es que el código de la tabla se incluirá mediante un sequence, por lo que tendremos que incluir el nombre del mismo en la configuración del fichero.

```
<id name="codigoLibros" type="java.lang.Long">
        <column name="CODIGO_LIBROS" precision="11" scale="0" />
        <generator class="sequence" >
              <param name="sequence">codigo_libros</param>
        </generator>
</id>
```

Otro de los cambios importantes es la introducción de referencias a otras tablas. En el cual tendremos que incluir el nombre del atributo, la dirección donde se encuentra el objeto con el que se tiene la relación y finalmente incluir el dato que crea la relación en la base de datos, a continuación mostramos un ejemplo del código necesario:

```
<many-to-one name="escritores" class="com.rivas.hibernate.Escritores"
fetch="select">
   <column name="CODIGO_ESCRITOR_LI" precision="11" scale="0" />
</many-to-one>
```

Es muy importante incluir la relación “many-to-one” para indicar que muchos de los libros creados pueden pertenecer a 1 solo escritor.

4.4.2.3. Paquete hibernate.dao

En este paquete se contendrá a los DAO y los DAO extendidos de todas las clases. Dichos DAO contienen operaciones básicas para trabajar entre la aplicación y la Base de Datos como pueden ser inserciones, modificaciones, consultas o borrados. A continuación y siguiendo el ejemplo de antes comentaremos el funcionamiento de uno de los DAO presentes ya que todos son extremadamente similares.

LibrosDAO.java

Todos los DAO extenderán de "HibernateDaoSupport" para garantizar que se crean y gestionan las sesiones de Hibernate. Normalmente los DAO contienen una gran cantidad de métodos para interactuar con la Base de Datos. A continuación se comentarán los métodos más interesantes y más usados:

- public void save(Libros instance): Método que recibe por parámetro un objeto libro y lo guarda en la Base de Datos.
- public void delete(Libros instance): Recibe un libro por parámetro y lo elimina de la Base de Datos. Normalmente teniendo el ID libro es suficiente para la eliminación, no es necesario tener todos los atributos para la eliminación.
- public Libros findById(java.lang.Long id): Se le indica un ID y busca el libro correspondiente al mismo.
- public List findByNombreLibro(Object nombreLibro): Recibe un nombre de libro y devuelve una lista con los posibles resultados. A diferencia del método anterior, el ID es único pero el nombre libro no, por lo que puede haber múltiples resultados
- public List findAll(): Método que ejecuta el clásico "SELECT * ". Devuelve un listado completo de todos los libros de la tabla Libros.
- public void attachDirty(Libros instance): Recibe un objeto libro. Si el objeto ya existe (existe el ID) en la Base de Datos lo modifica, si no existe crea una nueva entrada en la Base de Datos con el nuevo libro.

LibrosDAO_EXT.java

Es una clase hija de LibrosDAO.java y en ella incluiremos otro tipo de sentencias no tan genéricas como las mostradas en la anterior clase.

Gracias a la clase "DetachedCriteria" de Hibernate podemos realizar consultas más elaboradas sobre la base de datos como pueden ser Join's a otras tablas o poner restricciones a las búsquedas. A continuación mostraremos los métodos presentes en la clase:

- public List<Libros> consultar_TODO_by_escritor(Escritores esc): Este método recibe un objeto escritor y devuelve un listado de todos los libros escritos por el escritor indicado.
- public List<Libros> consultar_TODO_by_libro(): método que hace una consulta de todos los libros pero haciendo sentencias Join con cada una de las tablas con las que tiene relaciones. De esta forma recuperaremos los objetos completos de la relación, en vez del código que las relaciona.

4.4.2.4. Paquete hibernate.modelo

Este paquete contendrá todas las fachadas, es decir, contendrá todas las clases de unión entre la gestión de la Base de Datos (el paquete completo de Hibernate más la base de datos) y la aplicación. Estos métodos serán llamados desde la vista o el controlador de la aplicación y ellas a su vez, llamarán a los respectivos DAO para que realicen las diferentes acciones sobre la Base de Datos.

Estas clases son llamadas mediante el uso de Spring, por lo que cada una de ellas tendrá que tener una interfaz propia que incluirá todos los métodos a los cuales queremos acceder desde el exterior de la misma clase.

De igual modo que en los DAO, las fachadas son muy similares entre sí, por lo que tan solo comentaremos la fachada de los libros.

Gestion_Libro.java

Esta clase implementa su propia interfaz IGestion_Libro.java para el funcionamiento de Spring.

Como ya se dijo anteriormente esta clase es una intermediaria entre la aplicación en sí y los DAO por lo que estará compuesta por tantos métodos como métodos del DAO y el DAO_EXT queramos acceder.

De tal forma que crearemos una instancia de la clase LibrosDAO y haremos las llamadas necesarias a las funciones que componen la clase LibrosDAO o LibrosDAO_EXT.

Para facilitar más el trabajo posteriormente los métodos que creemos para la fachada les pondremos nombres significativos y fácilmente identificables. De esta forma el método de la fachada que llame al método "attachDirty(mi_libro);" del DAO lo llamaremos "modificar_Libro (Libros mi_libro)" al cual le pasaremos por parámetro el libro a modificar.

4.4.2.5. Paquete Imágenes

Este paquete contiene todos los documentos gráficos que se usan en la aplicación. Tiene 2 sub-paquetes, en uno de ellos guardamos las imágenes 16x16 que se asignan a los JMenu y el otro es donde ese guarda las portadas de los libros que se insertan.

4.4.2.6. Paquete spring

Acceso_ApplicationContext.java

Esta clase será la encargada de relacionar la aplicación con el núcleo de Spring mediante métodos estáticos para poder acceder al applicationcontext en cualquier

lugar de la aplicación. Contiene un único atributo, ctx, de tipo ApplicationContext que representa a esta factoría de beans, y tres métodos, que detallamos a continuación:

- Void setApplicationContext(ApplicationContext ctx): método heredado del interfaz ApplicationContextAware, cuya misión consiste en dar a conocer el applicationcontext al crear los beans.
- ApplicationContext getContextoSpring(): usado para acceder al webapplicationcontext en cualquier punto de la aplicación.
- Object getBean(String objeto): que es el encargado de la obtención directa del beans u objeto que vamos a crear.

ApplicationContext.xml

Como ya se explicó en el apartado 4. Toma de contacto, este fichero sirve de descriptor para el framework Spring. En él se definirán los Beans junto a su nombre y ruta de la clase que representa.

Spring tiene facilidades para implementar en él Hibernate, por lo que este fichero tiene información que el comentado en el punto 3.2.1 no tiene. EL primero de los parámetros nuevos a incluir es la conexión de Hibernate en la que tenemos que incluirle la ruta del fichero de configuración de Hibernate. El siguiente código muestra la forma de hacerlo.

```
<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
      <property name="configLocation">
            <value>
                  classpath:/com/rivas/hibernate/hibernate.cfg.xml
            </value>
      </property>
</bean>
```

La última parte importante a incluir son las transacciones de Hibernate, que sirven para especificar en qué momento los datos de Hibernate pasan de los objetos Java a las entidades de la base de datos. A continuación mostramos un código de ejemplo que más tarde explicaremos:

```
<!-- DEFINICION DE TRANSACCIONES EN HIBERNATE -->
   <!-- GESTOR TRANSACCIONAL -->
```

```
    <bean id="transactionManager"

    class="org.springframework.orm.hibernate3.HibernateTransactionManager"
/>
    <!-- DEFINICION TRANSACCION PARA LIBROS -->
    <aop:config>
          <aop:pointcut
                expression="execution (*
com.rivas.hibernate.modelo.Gestion_Libro.* (..))"
                id="puntos_cortelibro" />
          <aop:advisor advice-ref="reglas_transaccionlibro"
                pointcut-ref="puntos_cortelibro" />
    </aop:config>
    <!-- DEFINICION DE REGLAS DE TRANSACCION  -->
    <tx:advice id="reglas_transaccionlibro">
          <tx:attributes><tx:method name="*_Libro" />
          </tx:attributes>
    </tx:advice>
```

Como se puede observar se crea el "transactionManager" para Hibernate pasándole la ruta de un fichero propio de Hibernate. Más tarde definimos la transacción en la cual decimos que todos los métodos acabados en "*_Libro" de la clase Gestion_Libro podrán realizar transacciones con la base de datos.

Tendremos que crear tantas definiciones como clases en las que queramos hacer transacciones en la base de datos.

Nota: Las sentencias de consulta (SELECT) no se consideran transacciones, tan solo se consideran transacciones las sentencias que modifican datos de la Base de Datos (INSERT, UPDATE o DELETE).

4.4.2.7. Paquete vista

Este paquete contendrá todas las ventanas necesarias para poder gestionar todos los casos de uso de la aplicación, así como todas las interfaces respectivas para poder llamar a las ventanas mediante Spring y todos los archivos properties de idiomatización de fichas ventanas.

En este paquete se tienen los siguientes sub-paquetes: configuración, escritores, libros, préstamos y usuarios. El funcionamiento del contenido de escritores, libros y

usuarios es muy similar por lo que tan solo explicaremos uno de ellos que será el sub-paquete libros por tener algo más de complejidad en algunos puntos.

A continuación estudiaremos el contenido y estructura de los sub-paquetes de la vista.

4.4.2.8. Paquete vista.configuración

Aquí se incluirán todo lo correspondiente a la vista de la configuración de idioma. Tendremos la clase Vista_configuracion.java la cual contiene la ventana de configuración, la interfaz que implementación la misma y los properties para la idiomatización de la ventana.

Vista_configuracion.java

La clase contiene un método principal llamado "crear_Interface(Ventana ventana)" que recibe la ventana en la cual se pintará toda la interfaz. Debido al uso de Spring es necesario que el constructor de la clase no se incluya ningún código, ya que Spring no llama a constructores de clase sino a métodos.

Esta clase contiene 3 RadioButton para cada uno de los idiomas (español, inglés y alemán). Al entrar estará seleccionado el RadioButton del cual tenemos cargado el idioma, de forma que si la aplicación en ese instante está en alemán el RadioButton seleccionado por defecto al entrar será alemán.

Cuando se seleccione uno de ellos se activa un evento (actionPerformed) por el que se cambiará el idioma en la clase "Config_Idioma.java" dependiendo de cuál de los RadioBotton fue seleccionado. Al salir de esta ventana usando el botón "salir" la aplicación será cambiada de idioma.

4.4.2.9. Paquete vista.libros

Contendrá toda la gestión completa de libros. La gestión estará dividida en 3 clases o ventanas diferenciadas. Todas las clases contendrán un método llamado "crear_Interface(Ventana ventana)" al cual llamaremos desde Spring por lo que tendrá que contener todo el código de creación de la ventana.

En este paquete también contendremos las interfaces y los ficheros de idiomatización de las 3 ventanas. A continuación comentaremos las 3 clases contenidas en el paquete:

Libros_Vista_Agregar.java

Como ya se dijo, la clase contendrá un método llamado "crear_Interface(Ventana ventana)", lo primero que hará será consultar el idioma actual para crear los objetos en el idioma seleccionado.

La clase mostrará un formulario completo para introducir todos los atributos de un nuevo libro. Dependiendo del atributo el objeto que lo describirá será diferente, siendo un el nombre un JTextField, el escritor un combo box en el que seleccionar uno de los escritores ya presentes en la Base de Datos, o un TextArea para la descripción.

Para cada uno de los combo Box presentes necesitaremos hacer una llamada a la Base de Datos para cargar una lista con los valores presentes. Como ya se explicó para Hacer esta acción tendremos que llamar al método correspondiente de la fachada (en caso de escritores será Gestion_Escritor.java) y ya ella se encargará de llamar a Hibernate.

Al dar al botón "Guardar" se creará un objeto Libro con todos los atributos incluidos en el formulario y llamando al método "agregar_libro(mi_libro)" de la fachada de libros se añadirá un nuevo libro a la Base de Datos.

Libros_Vista_Listado.java

Al igual que la clase anterior esta tendrá un método llamado crear_Interfaz. Este método hace muestra todos los libros presentes en la Base de Datos, usaremos el método "consultar_Todo_by_libro()" explicado anteriormente, de tal forma que cuando nos traigamos un objeto libro de la base de datos también nos traeremos los

objetos Idioma, Escritor, Género y Editorial del mismo para poder mostrar su información correctamente y no solo su código identificador.

Alternativamente, se tendrá un combo Box el cual muestra un listado de los escritores y al seleccionar uno de ellos se hará un filtrado de la lista de libros para mostrar únicamente los libros pertenecientes al escritor seleccionado

Libros_Vista_Modificar.java

Esta clase será la encargada de modificar o eliminar libros de la base de datos. Se mostrará un Combo box llamado "Elegir Libro a Modificar" que tendrá un listado completo de todos los libros.

Al seleccionar uno de los libros que aparecen se mostrará toda su información en pantalla y también aparecerán 2 botones nuevos "Eliminar Libro" y "Guardar Modificación". Toda la información mostrada será completamente editable.

Al dar a cualquiera de los 2 botones el proceso que ocurrirá será muy similar. Albos llamaran al método de la fachada libros "modificar_Libro(mi_libro);" en el caso del botón de modificar guardará las modificaciones realizadas en el libro. Y en el caso de eliminar se llamará al mismo método "modificar_Libro(mi_libro);" cambiando el atributo de eliminado a "SI".

4.4.2.10. Paquete vista.prestamos

Prestamos_Vista_Agregar.java

Mostraremos en esta clase 2 bombo box, en el primero de ellos estará cargada la lista completa de Alumnos y en el segundo un listado completo de libros. Al dar al botón "Guardar" Se creará una nueva entrada en la tabla préstamos con el libro y el usuario seleccionados. Al mismo tiempo el número de libros prestados por el usuario seleccionado incrementará en 1 y en número de copias disponible disminuirá 1 unidad.

En caso de que el usuario ya tenga 3 libros en posesión o que no se tengas copias del libro seleccionado el préstamo no se llevará a cabo.

Prestamos_Vista_Listado.java

Al igual que en el listado de libros explicado anteriormente se mostrará un listado completo de todos los préstamos. Haremos una llamada al método "consulta_Prestamos();" de la fachada de préstamos el cual nos devolverá un listado completo de los mismos y antes de mostrarlos realizaremos un filtrado para mostrar únicamente los préstamos que no fueron devueltos.

Prestamos_Vista_Devolver.java

En esta ventana se incluirá un combo box el cual mostrará la información completa de los préstamos que no han sido devueltos todavía. Al dar al botón "guardar devolución" el préstamos e marcará como devuelto, el usuario restará 1 en los libros prestados y el libro devuelto aumentará 1 el número de copias disponibles.

4.5. Pruebas

En este apartado realizaremos pruebas sobre la aplicación comprobando que se cumplan los casos de uso propuestos anteriormente.

Se realizará el proceso completo de cómo se tendría que manejar la aplicación correctamente. Primeramente crearemos un objeto escritor para luego crear un libro y asignarle como escritor el objeto escritor creado anteriormente. Seguidamente se creará un usuario o alumno y finalmente se realizará un préstamo utilizando el libro y el usuario creados.

En todo este proceso se comprobará que todos los objetos creados aparecen en el listado de los mismos y pueden ser modificados y/o eliminados.

4.5.1. Pruebas sobre Idiomatización

A continuación mostraremos las pruebas de la idiomatización de la aplicación. Mostraremos las diferentes vistas dependiendo del idioma elegido en el apartado "Archivo→Configuración Idioma".

En la siguiente imagen se muestra la aplicación nada más entrar en ella, actualmente el idioma es español.

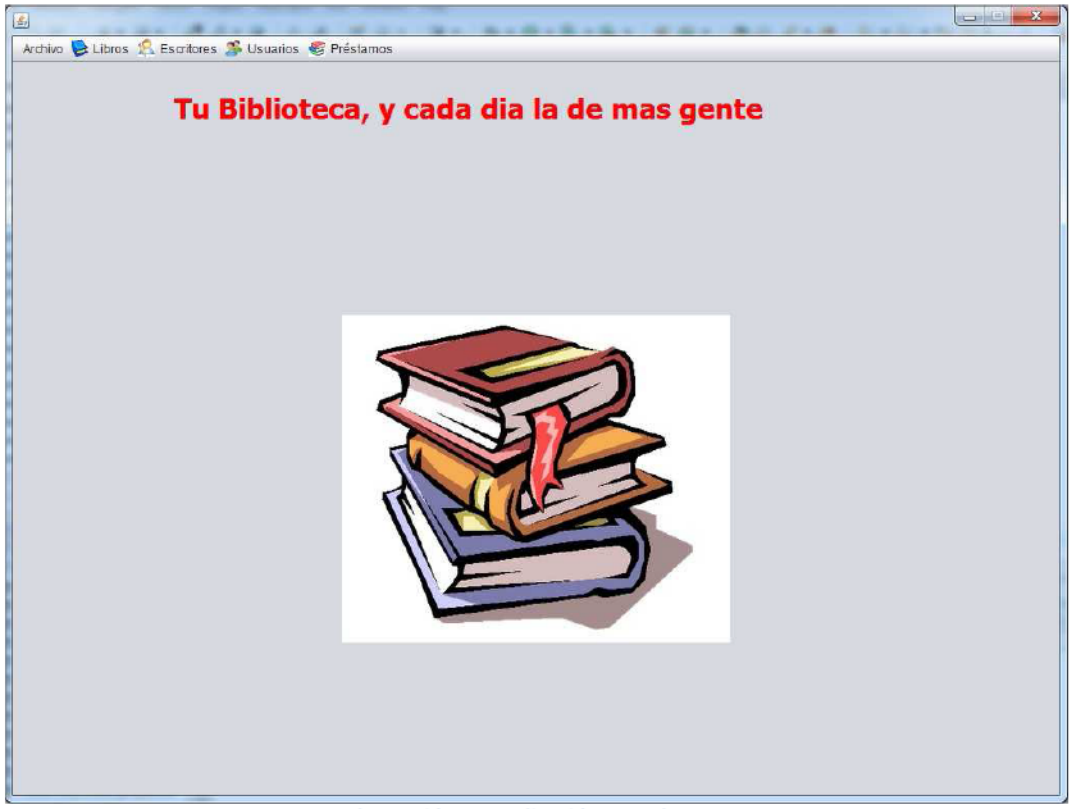


Ilustración 24. Aplicación completa

Accederemos a la parte de la gestión del idioma y nos aparecerá la siguiente ventana donde podremos elegir entre los diferentes idiomas de la aplicación.

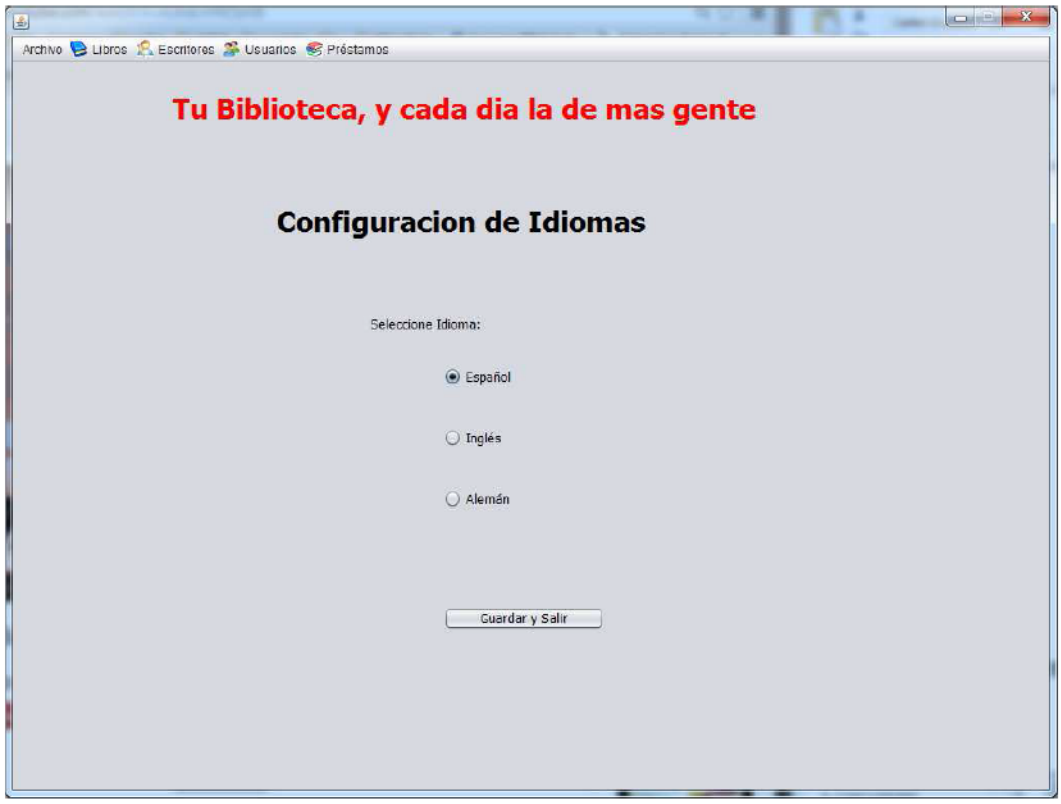


Ilustración 25. Vista configuración de Idiomas

Como se puede observar el idioma en este momento es “Español”. Se selecciona el idioma “Inglés” y al volver a la página principal podemos comprobar que todos los datos de la aplicación cambiaron de idioma.

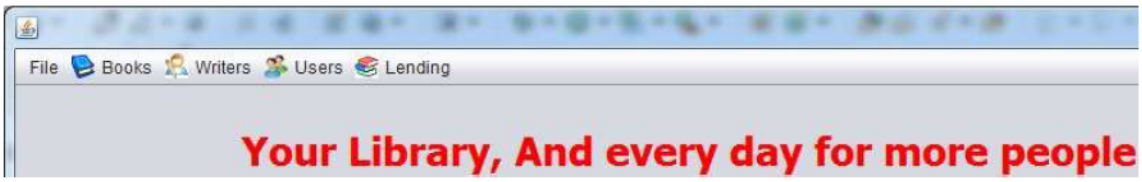


Ilustración 26. Aplicación en inglés

Hacemos el mismo proceso pero esta vez seleccionando el idioma “Alemán” y comprobamos que los datos se cargan correctamente en alemán.

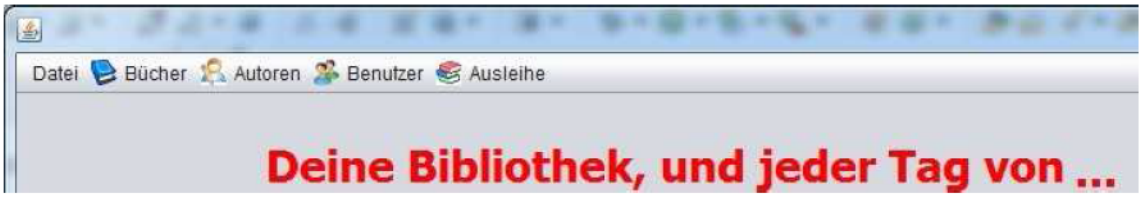


Ilustración 27. Aplicación en alemán

La última comprobación que haremos será cerrar nuestra aplicación y al volver a iniciarla comprobar que los menús nos los carga en último idioma que se seleccionó. Esto es gracias a un fichero de persistencia que se incluye en la aplicación donde se guarda y se carga el idioma elegido.

En este ejemplo tan solo se muestra la ventana inicial de la aplicación, pero el resto de las ventanas también estan idiomatizadas.

4.5.2. Pruebas sobre Escritores

4.5.2.1. Agregar Escritor

Para comenzar nuestras pruebas crearemos un escritor, en este caso crearemos a la escritor “Miguel de Cervantes Saavedra”, escritor del comodísimo “Don Quijote de la Mancha”, libro que agregaremos más adelante a nuestra aplicación. Para crear un escritor nuevo tenemos que acceder desde la ventana inicial a “Escritores→Agregar Escritor”.

En la siguiente imagen observamos la creación de un escritor y el mensaje de la aplicación que nos indica que la creación se llevó a cabo correctamente.

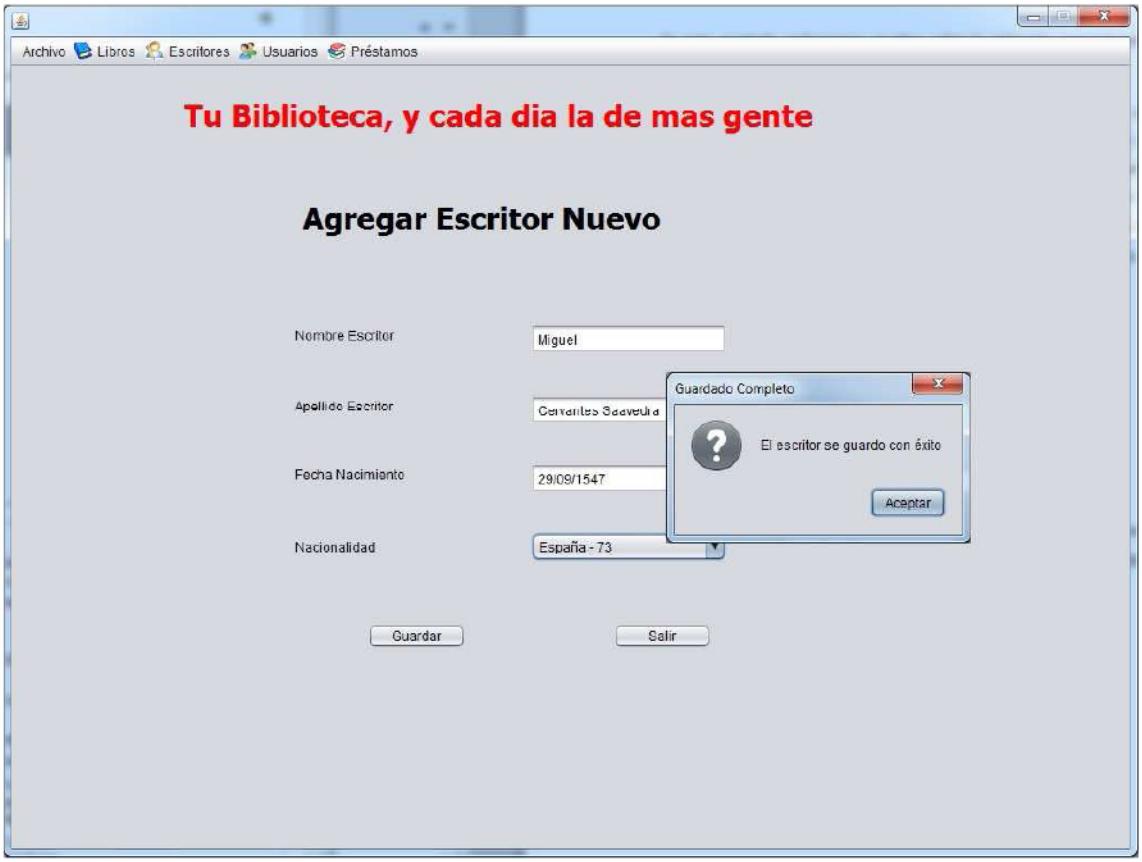


Ilustración 28. Vista agregar escritor

4.5.2.2. *Listado de Escritores*

Una vez agregado nuestro escritor accedemos a la vista del listado de escritores de la siguiente forma “Escritores→Listado de Escritores”. En la siguiente imagen podemos comprobar que el escritor creado en el paso anterior se muestra correctamente en el listado de los escritores que contiene la Base de Datos.

Ilustración 29. Vista lista escritores

4.5.2.3. *Modificar Escritor*

Una vez hemos comprobado que el escritor que hemos introducido se agregó correctamente a la base de datos vamos a proceder a modificar alguno de sus campos.

Se procederá a cambiar el apellido del autor creado de "Cervantes Saavedra" a "de Cervantes Saavedra". Una vez en la vista de modificar escritor seleccionaremos en el combo box superior a nuestro escritor y se cargarán los datos del mismo en el formulario. Cambiamos el apellido y le damos al botón "Guardar Modificación".

En la siguiente imagen vemos el resultado.

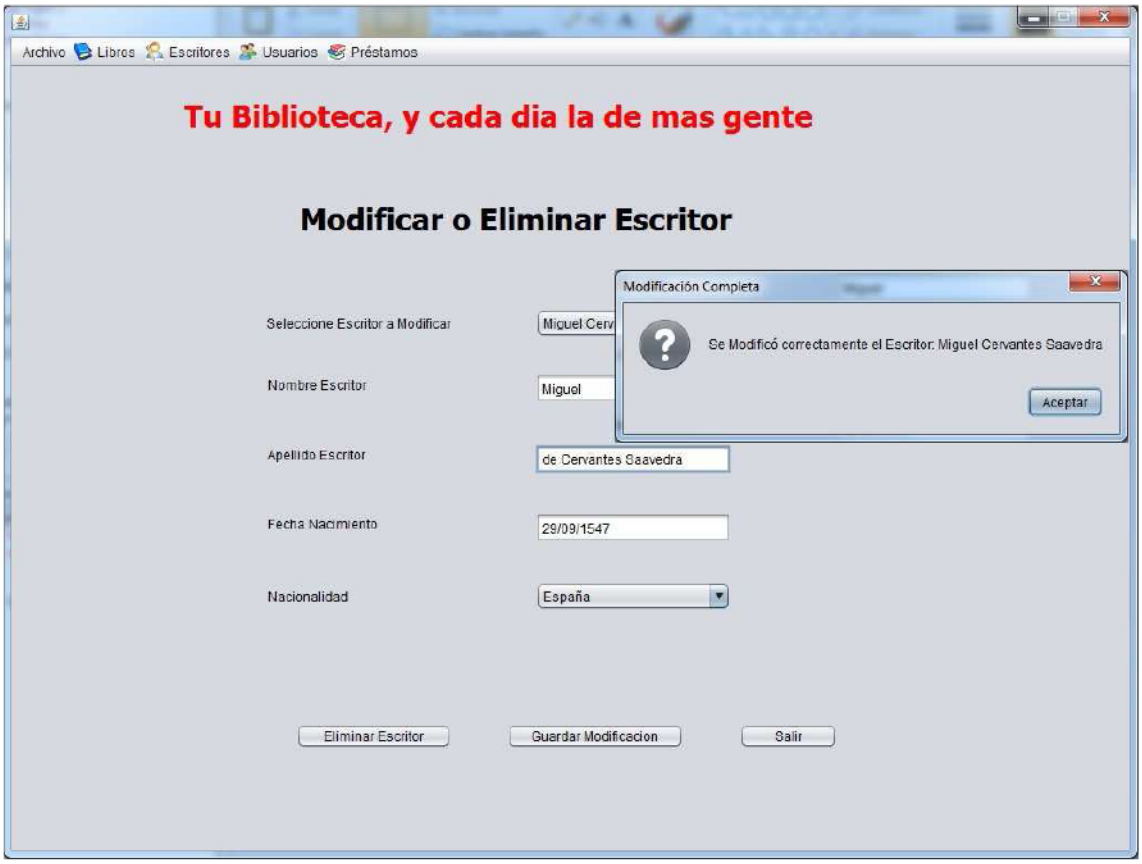


Ilustración 30. Vista modificar escritor

4.5.2.4. *Eliminar Escritor*

La última de las pruebas que realizaremos sobre los escritores será eliminar un escritor creado. Recordamos que la eliminación consiste en cambiar el campo "eliminado" de la base de datos.

Para este caso se creó un escritor de prueba llamado "pepeeee fffff" el cual eliminaremos mediante el botón "Eliminar escritor".

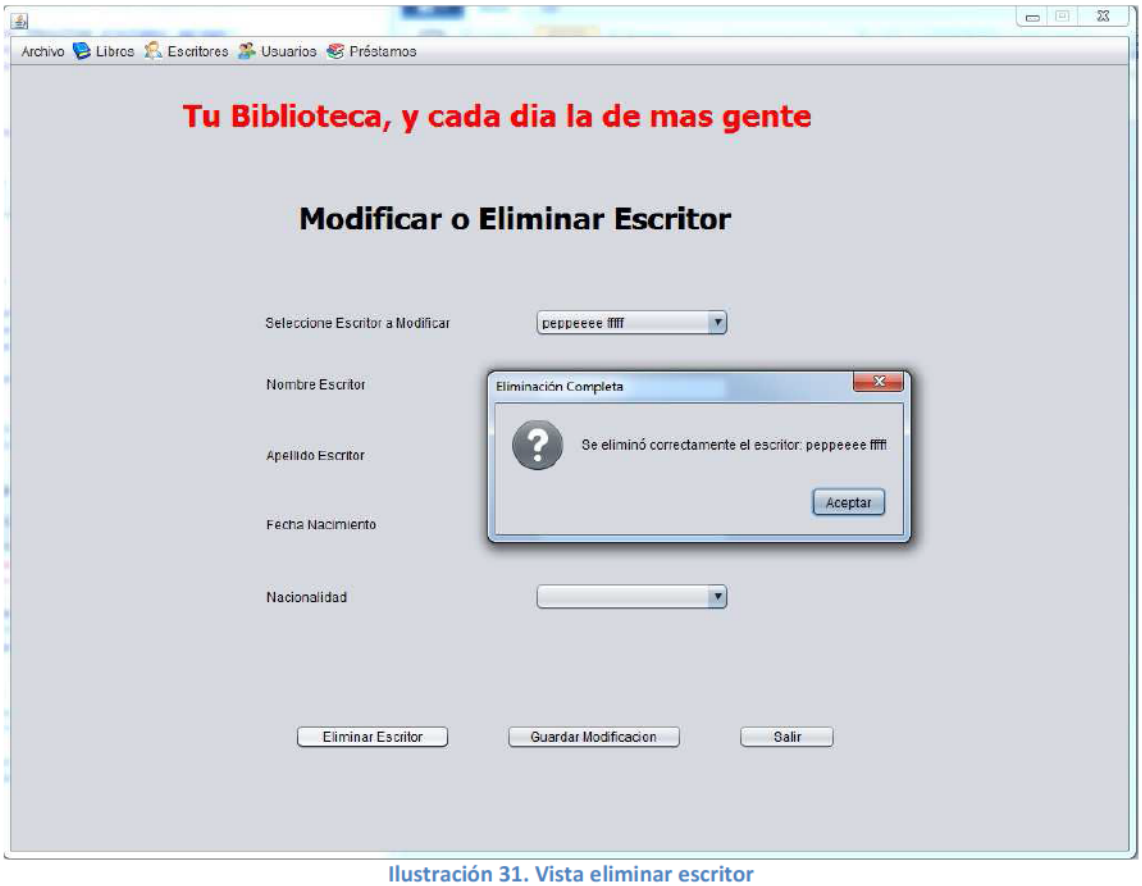


Ilustración 31. Vista eliminar escritor

4.5.3. Pruebas sobre Libros

4.5.3.1. Agregar Libros

Una vez creado nuestro escritor ahora se creará el libro “Don Quijote de la Mancha” al cual le asignaremos el escritor que creamos antes como autor del mismo.

Lo primero será acceder a la vista de agregar libro “Libros → Agregar Libro”. Una vez en ella rellenamos todos los campos del formulario y le damos al botón “Guardar” el resultado aparece en la siguiente imagen.

Ilustración 32. Vista agregar libro

4.5.3.2. *Listado de Libros*

Una vez insertado el libro iremos a la vista de listado de libros "Libros → Listado de Libros" en la cual podemos comprobar que el libro se agregó correctamente y se muestra su información.

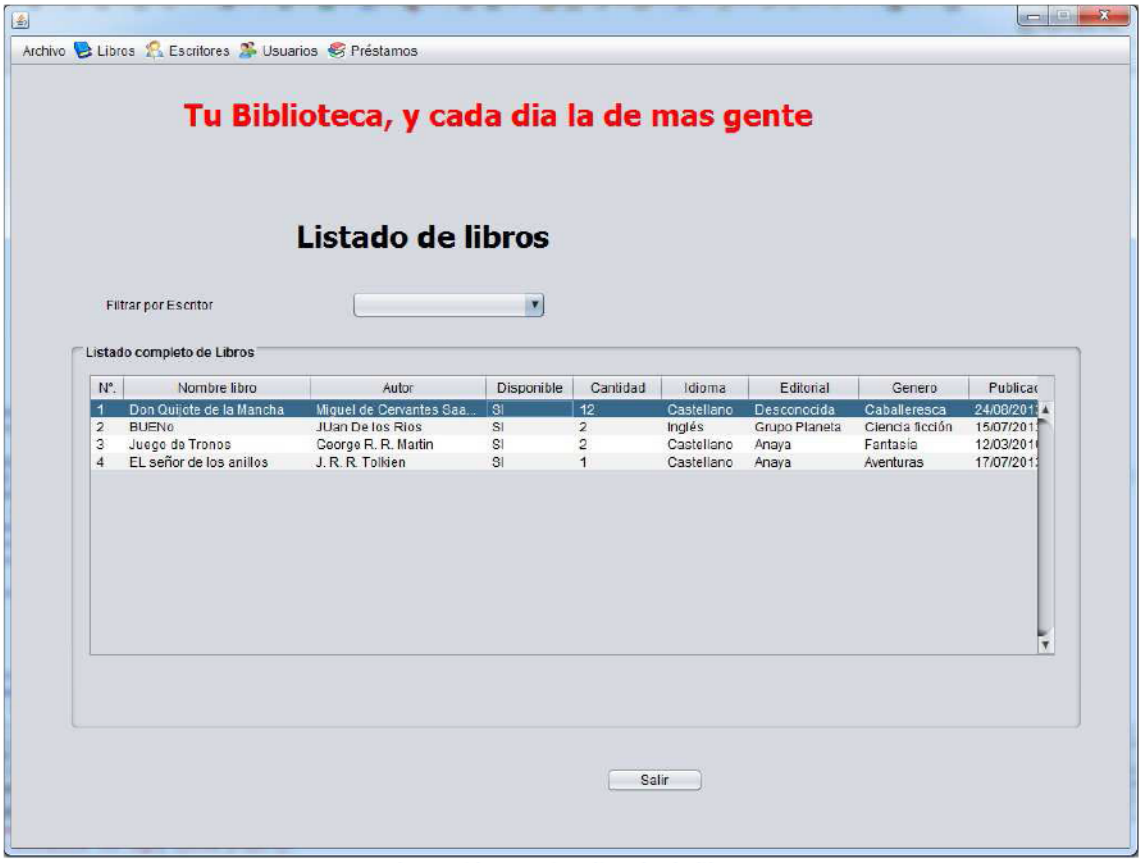


Ilustración 33. Vista listado de libros

En la imagen se observa que encima de la tabla con la información de los libros hay un combo box. Este combo box posee el listado de todos los escritores, de tal forma que si se selecciona uno de ellos se realzará un filtrado de libros por escritor y tan solo se mostrarán los libros de ese escritor.

4.5.3.3. *Modificar Libros*

Se decide modificar algunos de los atributos del libro recientemente creado, para ello iremos a la vista de modificar libro "Libros → Modificar o Eliminar Libros".

Como vemos el primero de los combo box es para elegir el libro a modificar, seleccionamos el nuestro y se carga automáticamente la información de este en el formulario. Decidimos cambiar los campos descripción y cantidad y le damos al botón "Guardar Modificación", el cual se encargará de guardar en la base de datos los cambios.

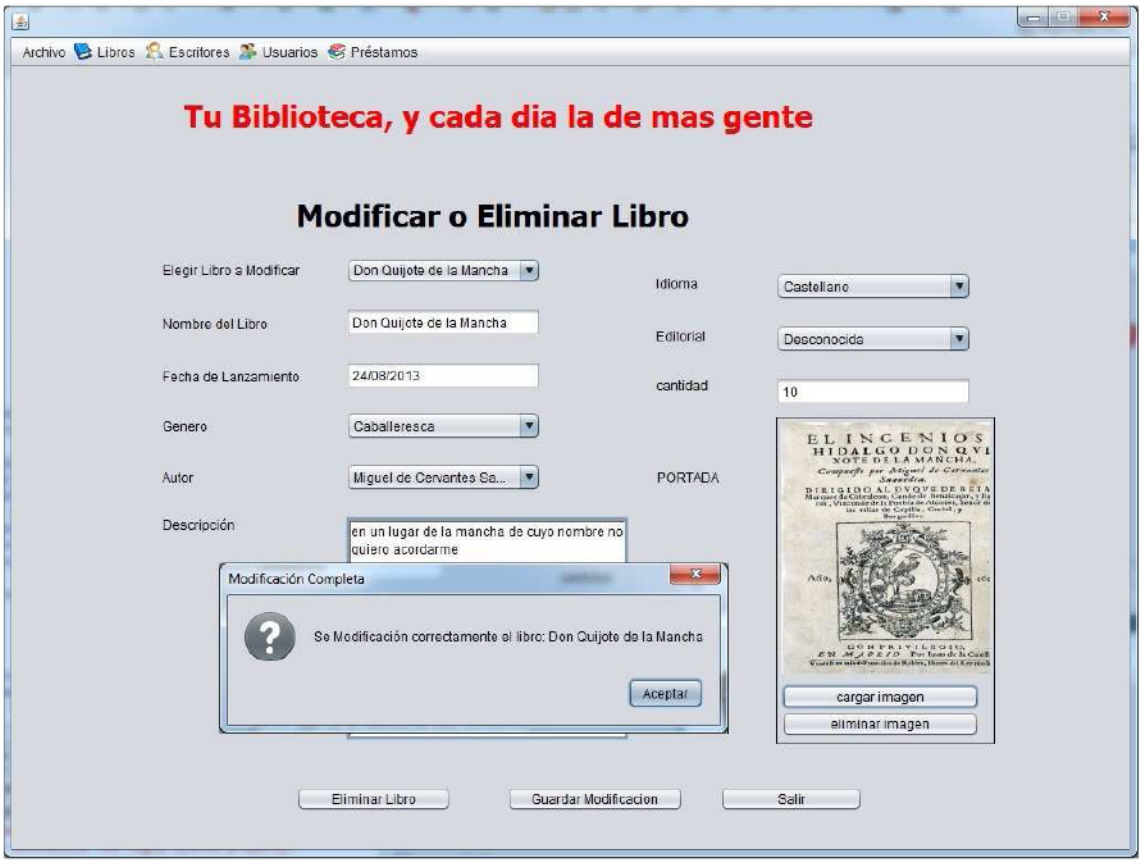


Ilustración 34.Vista modificar libro

4.5.3.4. *Eliminar Libros*

Para finalizar con las pruebas sobre libros vamos a eliminar uno de la base de datos, para ello vamos a "Libros → Modificar o Eliminar Libros".

En el combo box superior seleccionamos el libro que queremos eliminar en nuestro caso será un libro que se creó de prueba con el nombre "Bueno". Una vez seleccionado el libro le damos al botón "Eliminar Libro" y el resultado es el que aparece en la siguiente imagen.

Ilustración 35. Vista eliminar libro

4.5.4. Pruebas sobre Usuarios

4.5.4.1. Agregar Usuarios

En los casos de uso se especificaba que se quería una gestión completa de usuarios y es lo que vamos a comprobar en los siguientes apartados.

Primero empezaremos con la creación de un nuevo usuario, para ello desde el menú de inicio vamos a "Usuarios → agregar usuarios" y accederemos a la vista de agregar usuario nuevo en el cual se nos mostrará un formulario.

Rellenáramos el formulario que se nos presenta, en este caso crearemos un usuario con nombre "Luis" y apellido "Martínez". El resultado de la inserción se muestra en la siguiente imagen.

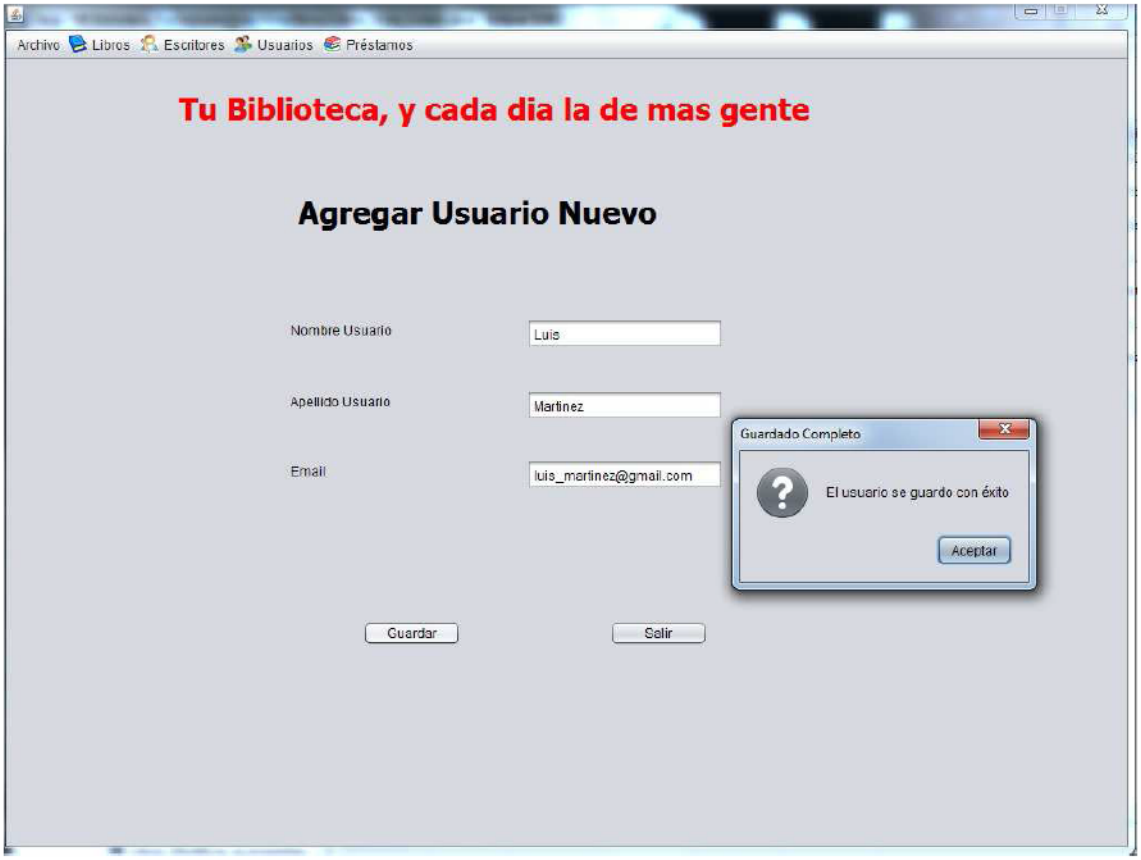


Ilustración 36. Vista agregar usuarios

4.5.4.2. *Listado de Usuarios*

Accedemos a la lista de usuarios "Usuarios → Listado de usuarios" para comprobar si el usuario recién creado se muestra en dicha lista.

Como comprobamos el usuario con recién creado de nombre y apellidos "Luis Martínez" se muestra correctamente.

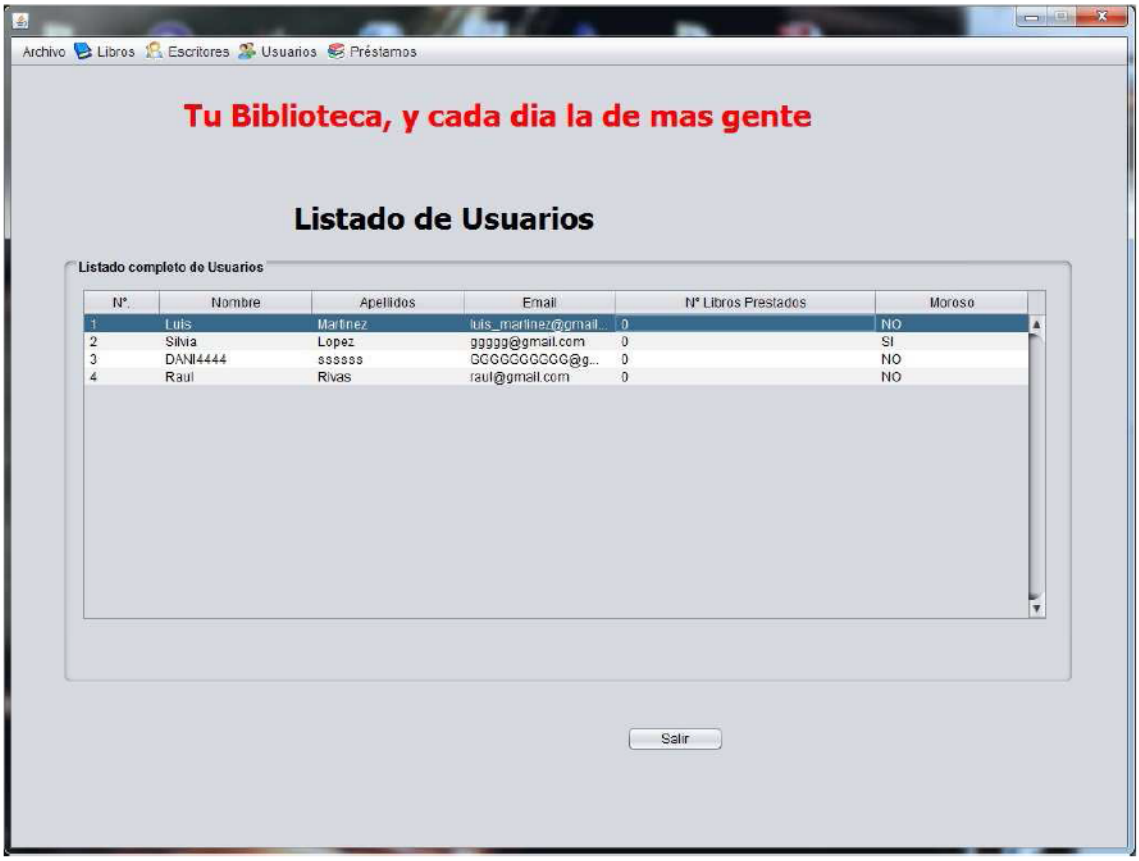


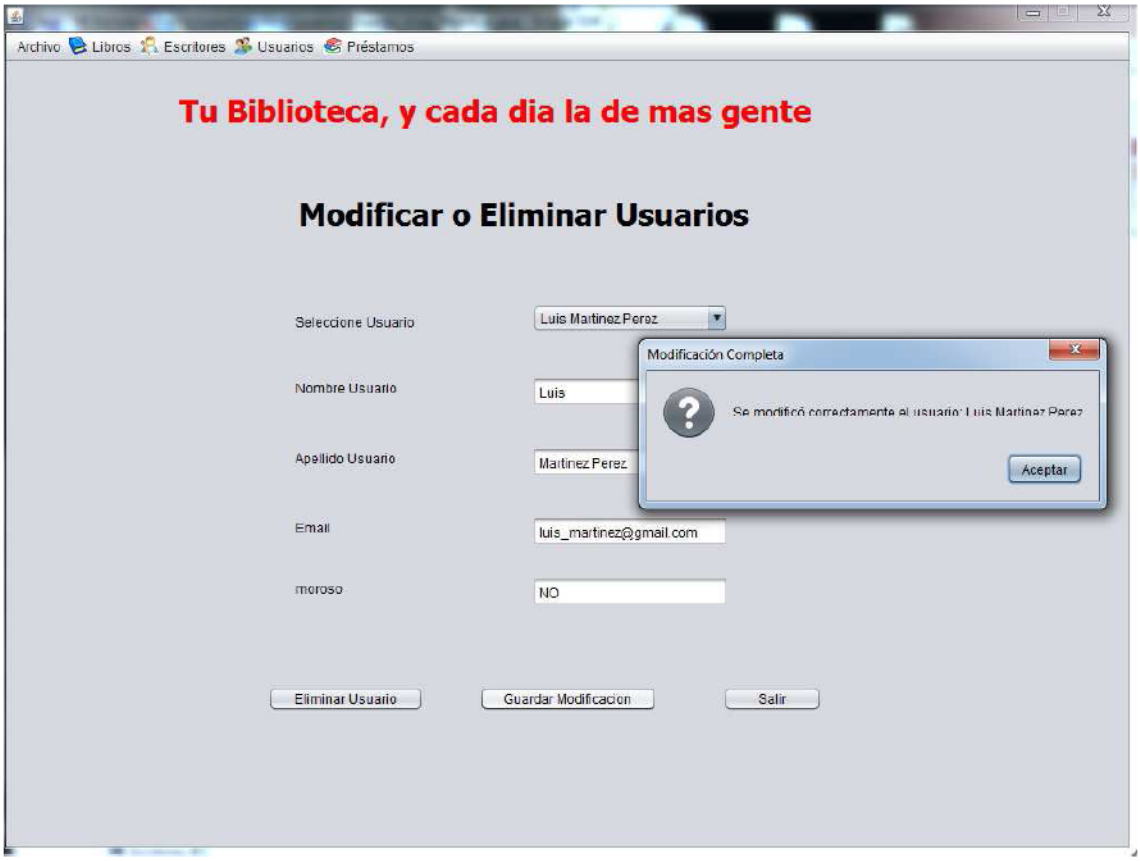
Ilustración 37. Vista listado libros

4.5.4.3. *Modificar Usuarios*

A continuación modificaremos el usuario recién creado, para ello vamos a "Usuarios → Modificar o Eliminar usuarios".

Se ha decidido modificar el campo apellido para que contenga ambos apellidos. Para ello en la ventana de modificación de usuarios se selecciona el usuario que se quiere modificar para mostrar su información y modificarla.

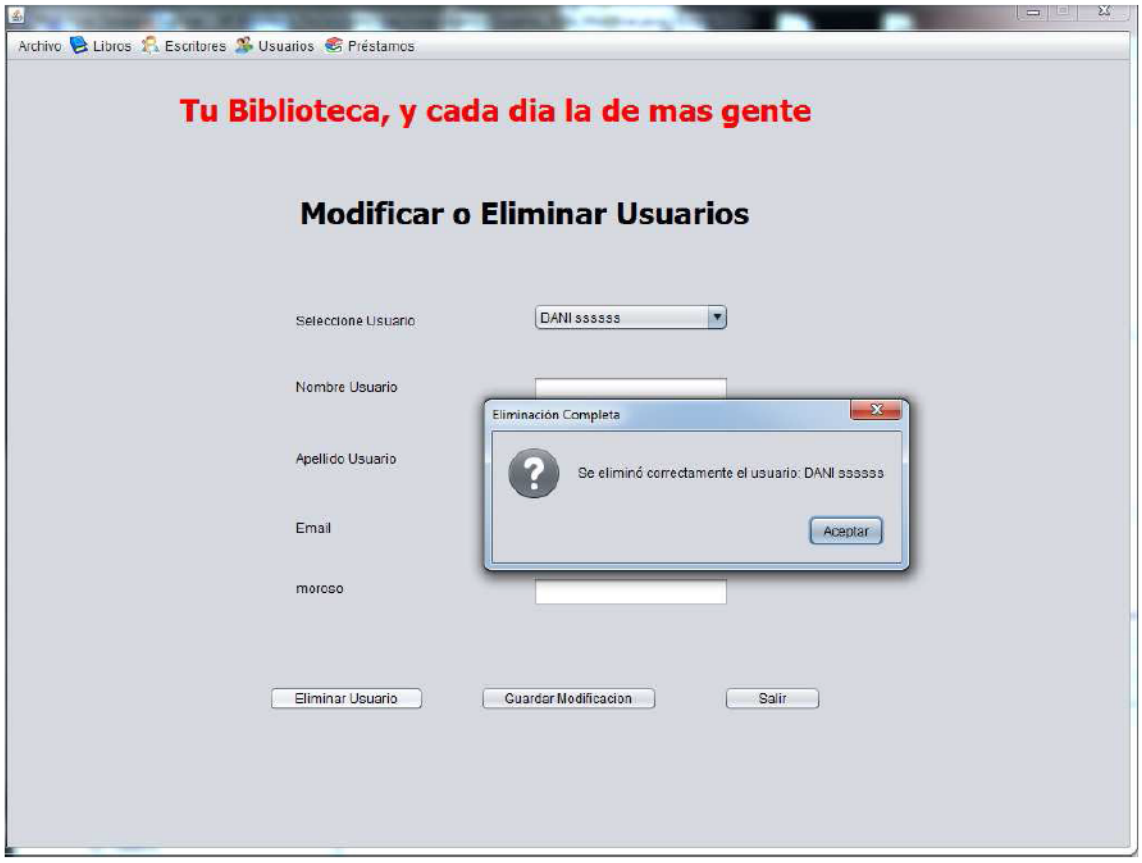
Al darle al botón "Guardar Modificación" la información del usuario presente en la base de datos será cambiada por la nueva. El resultado de la modificación se muestra en la siguiente imagen.



4.5.4.4. Eliminar Usuarios

Para finalizar con las pruebas de usuarios se eliminará un usuario de la base de datos, para ello accederemos a la ventana "Usuarios → Modificar o Eliminar usuarios" desde la ventana inicial de la aplicación.

Seleccionamos en el combo box el usuario que se quiere eliminar, en este caso será un usuario de prueba que se creó con nombre "DANI". Una vez seleccionado le damos al botón "Eliminar Usuario" y el usuario es eliminado de la aplicación.



4.5.5. Pruebas sobre Préstamos

4.5.5.1. Agregar Préstamos

Finalmente llegamos a la parte de préstamos. Usaremos los registros creados anteriormente para realizar nuestro préstamo, de tal forma que crearemos un préstamo con el libro "Don Quijote de la mancha" y con el usuario "Luis Martínez Perez".

Para realizar esta acción iremos desde la ventana principal a "Préstamos → Agregar Préstamo". Una vez allí seleccionaremos el libro y el usuario elegidos en sendos combo box. Para realizar el préstamo es imprescindible que el usuario tenga 2 o menos libros prestados y que el libro esté disponible, en caso contrario saltará un erro y no se llevara a cabo el préstamo. El resultado del préstamo se aprecia en la siguiente imagen.

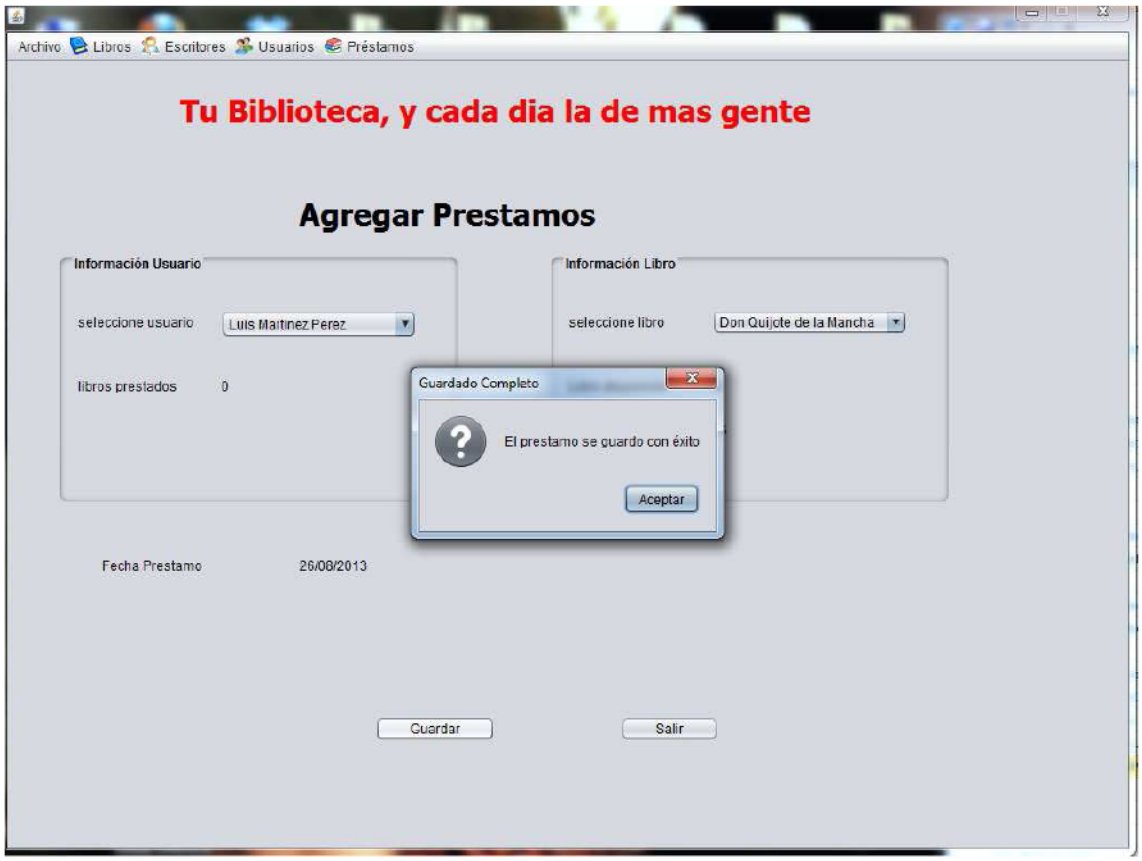


Ilustración 38. Vista agregar préstamos

Hay que destacar que a la hora de hacer el préstamo se crea el préstamos y aparte al usuario seleccionado se le suma 1 en el campo de libros prestados al libros se le resta uno en su cantidad disponible.

4.5.5.2. Listado de Préstamos

Una vez creado el préstamo vamos a comprobar que se muestre en el listado junto al resto de los préstamos. Para acceder a ella desde ventana principal seguimos la siguiente secuencia "Préstamos → Listado de Préstamo".

En este listado no se muestran los préstamos ya devueltos, solo se muestran los pendientes por devolver.

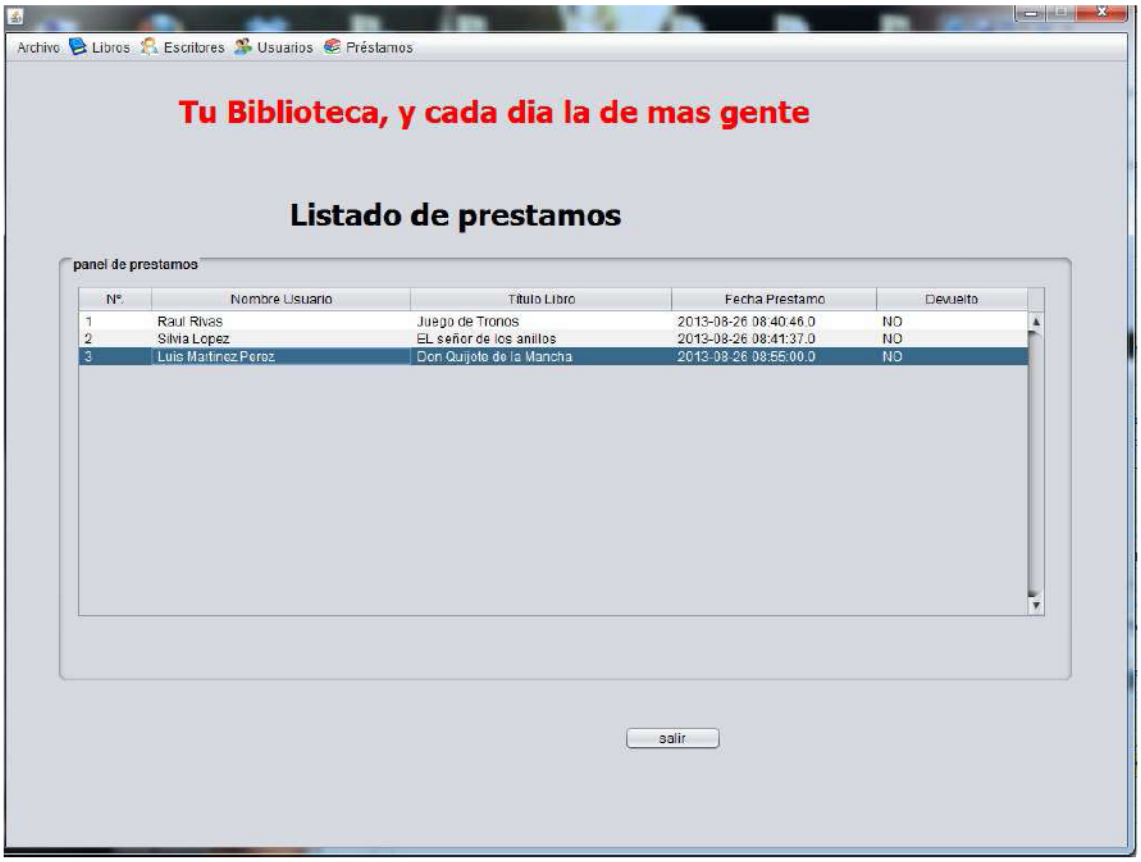


Ilustración 39. Vista listado préstamos

Como se aprecia en la imagen el préstamo se muestra correctamente.

4.5.5.3. *Devolver Préstamos*

Finalmente lo único que nos queda sería devolver el préstamo. Accedemos al menú de devolución desde "Préstamos → Devolver Préstamo".

El diálogo con esta ventana es muy simple. Se muestra un combo box con la información de los préstamos incluyendo usuario y libro. Seleccionamos el que se quiera devolver y dando al botón "guardar devolución" el sistema se carga de marcar como devuelto el préstamo, decrementar el número de libros del usuario y aumentar el número de copias del libro disponibles.

EL resultado se muestra en la siguiente imagen.

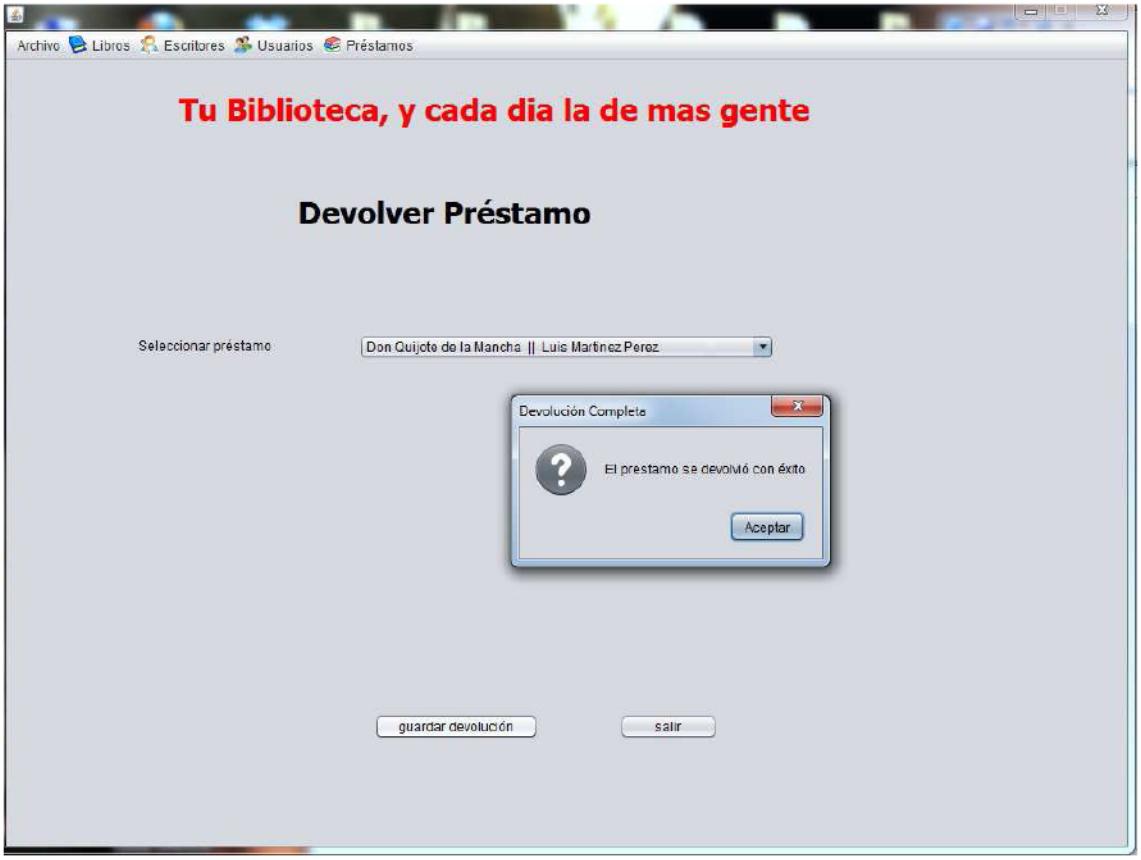


Ilustración 40. Vista devolver préstamo

5. FUTUROS TRABAJOS O MEJORAS

A continuación se expondrán diferentes casos de mejora o ampliación para la prueba de concepto, los cuales podrían formar parte de un futuro proyecto:

- Base de datos externa: Actualmente para que el programa funcione correctamente la máquina debe tener instalada y configurada la base de datos de la aplicación. Como futuro trabajo se propone albergar la base de datos en un servidor de Internet (a poder ser gratuito) de tal forma que cualquier computador con acceso a internet pudiese ejecutar la aplicación correctamente.

  En esta migración de la base de datos se tendría que cambiar el fichero hibernate.cnf.xml de tal forma que el campo "connection.url" apunte al servidor creado.

- Servicio mail: Muy frecuentemente los alumnos olvidan las fechas de devolución de los libros. Con esta opción se podrían enviar correos electrónicos a los alumnos (el correo es un campo obligatorio en la creación de usuarios) unos días antes del cumplimento del préstamo de modo de recordatorio.

  De esta forma se podrían notificar otros muchos casos vía e-mail un préstamo correcto o que hay disponibles libros nuevos en la biblioteca.

- Servicio de reserva de libros: Un servicio muy interesante a poner en funcionamiento sería un servicio de reservas de libros en caso de estar interesado en un libro pero que en ese momento no se dispongan de copias disponibles. De esta forma cuando el libro en cuestión quede libre el alumno tendrá X días para ir a recogerlo. El aviso de que el libro queda libre sería muy interesante realizarlo con el servicio mail comentado anteriormente.

- Gestión automática de usuarios morosos: Actualmente en la aplicación al realizar un préstamo queda registrada la fecha en la que fue realizado tal préstamo, pero por el momento no se contempla una fecha de devolución para tal préstamo, ni hay una gestión automática de ello.

  Se propone crear una gestión automática de los préstamos, de tal forma que si se sobrepasa la fecha de devolución (que sería un campo nuevo a crear en la base de datos) de un préstamo por parte del usuario este pase automáticamente a estado de moroso y que no se le permitan nuevos préstamos hasta que no sea devuelto dicho préstamo.

- Validación de campos con expresiones regulares: En este momento la aplicación no valida si los campos introducidos en los formularios son correcto o no, tan solo comprueba en algunos casos críticos que todos los datos del formulario estén cumplimentados.

  Como mejora de la aplicación se propone implementar validaciones de todos los campos mediante expresiones regulares. De esta forma no se tendría que confiar en que el usuario que maneja la aplicación introduzca los valores correctamente, sino que no se le permite introducir valores erróneos.

6. CONCLUSIONES

Tras la conclusión del desarrollo de las tecnologías y creación de la prueba de concepto, es hora de valorar resultados y sacar conclusiones. Tal y como se recogía en el apartado inicial de este documento, el objetivo principal del presente TFG era acercar a los lectores el framework Spring y el ORM Hibernate y de esta forma incitar y promover el uso de estas 2 tecnologías tan útiles y extendidas en los entornos de desarrollo Java.

Se han analizado todos los puntos fuertes tanto de Hibernate como de Spring y se han detallado en profundidad junto con ejemplos de código. De esta forma cualquier programador con poca experiencia en Java puede comprender y utilizar ambas tecnologías.

Mediante la creación de la prueba de concepto del préstamo bibliotecario hemos demostrado la sencillez con la que se puede desarrollar una aplicación Java utilizando las tecnologías Hibernate y Spring en conjunto. Gracias a la prueba de concepto hemos podido profundizar más en algunos aspectos de ambas tecnologías como el "DetachedCriteria" de Hibernate para crear sentencias complejas a la base de datos o la gestión de transacciones de Hibernate usando Spring, las cuales no se trataban en la toma de contacto con las tecnologías.

Este TFG nos ha dado una idea real, con su tiempo y esfuerzo dedicado, de lo que conlleva la realización de un desarrollo de unas nuevas tecnologías en un proyecto serio y completo. Esto nos da una aproximación de que nos podremos encontrar en nuestro día a día en el futuro laboral como ingeniero informático.

Por tanto, se puede afirmar que el proyecto propuesto cubre con éxito las expectativas marcadas inicialmente de dar a conocer el manejo y uso de Hibernate y Spring.

7. APENDICES

7.1. Presupuesto:

Para realizar el cálculo del presupuesto del proyecto se han tomado en consideración diferentes factores los cuales enumeraremos y detallarán a continuación:

Hardware necesario

Tabla 17. Presupuesto, coste de hardware

Elemento	Coste por unidad	Nº unidades	Coste total
CPU para implantación	600,00 €	1	600,00 €
Ordenador personal para desarrollo	800,00€	1	800,00€

La CPU para implementación será el equipo del cliente en el cual se le instalará y configurará todo para el correcto funcionamiento de la aplicación final, por ese modo el coste de este equipo será íntegro.

Sin embargo al ordenador personal para desarrollo se le aplicará una amortización acorde con la duración del proyecto

Software necesario

Tabla 18. Presupuesto, coste de software

Elemento	Coste por unidad	Nº unidades	Coste total
Licencia Windows 7	164,00 €	1	164,00 €
Licencia paquete Microsoft Office 2010 (Word, Excel, Visio,..)	199,00€	1	199,00€

Myeclipse Standard	31,70 €/año	1	31,70 €/año
Otros software libres (apache, eclipse, …)	0	1	0

Al igual que en el punto anterior a estos productos se les aplicará una amortización en base a la duración del proyecto.

Cálculos de amortización HW y SW

A continuación calcularemos los precios del Hardware y Software listados anteriormente aplicándoles la amortización.

Se ha fijado que la duración del proyecto será de aproximadamente de 6 a 8 meses por lo que se aplicará un 20 del coste total de los productos si son del desarrollador y un 100 si los equipos o productos serán propiedad del cliente.

Al producto Myecplise por ser una licencia anual se le aplicará el 100 del precio.

Tabla 19. Presupuesto, amortización HW y SW

Elemento	**Coste completo**	**Amortización**	**Coste Amortizado**
CPU para implantación	600,00 €	100	600,00 €
Ordenador personal para desarrollo	800,00€	20	160,00€
Licencia Windows 7	164,00 €	20	32,80 €
Licencia paquete Microsoft Office 2010 (Word, Excel, Visio,..)	199,00€	20	39,80 €
Myeclipse Standard	31,70 €/año	100	31,70 €
Otros software libres	0	20	0
COSTE TOTAL HW y SW			**864,30 €**

Costes de Personal

A continuación mostraremos el coste del personal que colabora en el proyecto:

Tabla 20. Presupuesto, coste de personal

Persona	Coste por hora	Horas totales	Total
Consultor	20 €	480 h	9.000,00 €
Analista	15 €	160 h	2.400,00 €
TOTAL PERSONAL			**11.250,00 €**

En el proyecto trabajarán un analista que será el encargado de realizar requisitos de usuario para la aplicación y un consultor que se encargará de investigar en las tecnologías, hacer la toma de contacto con las mismas y finalmente hacer la aplicación y documentarla correctamente.

Costes de Transporte

A todos los miembros del se les da una ayuda de transporte de 4€ diarios tanto si usan su coche propio o transporte público para ir a trabajar.

Tabla 21. Presupuesto, coste transporte

Persona	Jornada de trabajo	Días trabajados	Ayuda diaria	Total
Consultor	8 h	60 días	4€	240,00 €
Analista	8 h	20 días	4€	80,00 €
TOTAL TRANSPORTE				**320,00 €**

Costes de Dietas

A todos los miembros del equipo que trabajen jornada completa se les dará una ayuda de comida para costear el comer fuera de casa. Esta ayuda será de 9 € diarios.

Tabla 22. Presupuesto, coste dietas

Persona	Jornada de trabajo	Días trabajados	Ayuda diaria	Total
Consultor	8 h	60 días	9€	540,00 €
Analista	8 h	20 días	9€	180,00 €
TOTAL DIETAS				**720,00 €**

Costes Totales

Unificando todos los costes desglosados anteriormente, obtendremos como resultado final el coste total del desarrollo de proyecto:

Tabla 23.Presupuesto, costes totales

Coste total de:	Total
Hardware y Software	864,30 €
Personal	11.250,00 €
Transporte	320,00 €
Dietas	720,00 €
COSTE TOTAL DEL PROYECTO	**13.154,30 €**

Dado que el proyecto ha sido desarrollado por la Universidad Carlos III de Madrid con fines de investigación y sin carácter comercial, no se aplican porcentajes extra por riesgos y beneficios, junto con los costes computados de facturación de personal

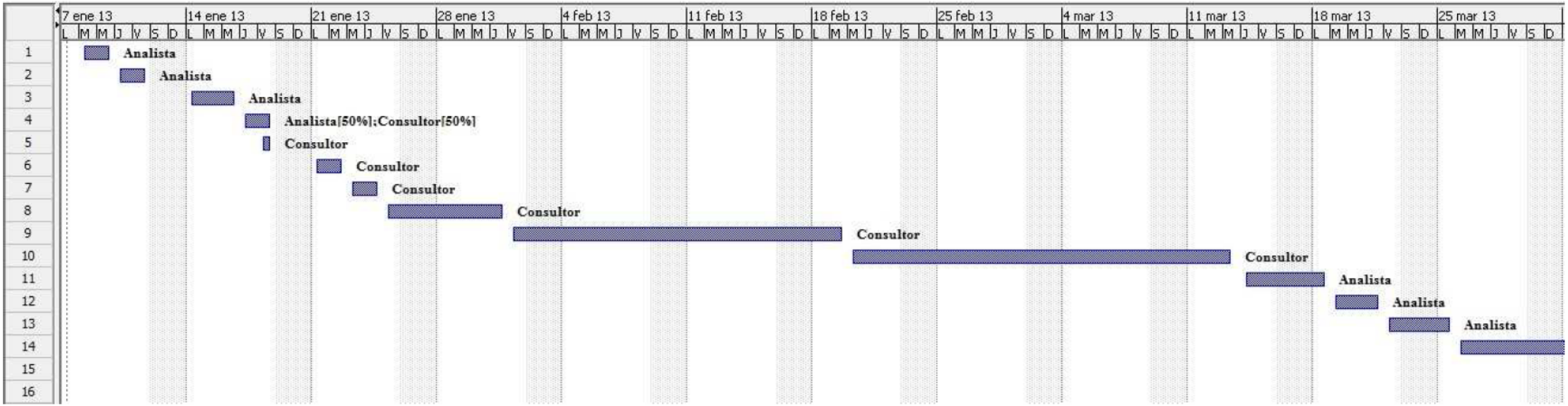
7.2. Planificación de Tareas:

A continuación mostraremos mediante un diagrama de Gantt la planificación de las principales tareas a realizar en el proyecto, el tiempo aproximado que llevó realizarlas y que persona fue la encargada de realizarla.

La planificación de tareas la compondrán una tabla resumen que mostraremos a continuación y el diagrama de Grantt correspondiente a la tabla.

Ilustración 41. Planificación de Tareas

	Nombre	Duración	Inicio	Terminado	Nombres del Recurso
1	Captura de requisitos de usuario con el cliente	2 days	8/01/13 8:00	9/01/13 17:00	Analista
2	Captura de casos de uso con el cliente	2 days	10/01/13 8:00	11/01/13 17:00	Analista
3	Redacción de requisitos y casos de uso	3 days	14/01/13 8:00	16/01/13 17:00	Analista
4	Traspaso de conocimientos (Analista - Consultor)	2 days	17/01/13 8:00	18/01/13 17:00	Analista[50%];Consultor[50...
5	Investigación Base de datos Oracle	1 day	18/01/13 8:00	18/01/13 17:00	Consultor
6	Investigación Leguguaje de Base de Datos SQL	2 days	21/01/13 8:00	22/01/13 17:00	Consultor
7	Investigación lenguaje de programación Java	2 days	23/01/13 8:00	24/01/13 17:00	Consultor
8	Investigación entorno gráfico Java Swing	5 days	25/01/13 8:00	31/01/13 17:00	Consultor
9	Investigación ORM Java Hibernate	13 days	1/02/13 8:00	19/02/13 17:00	Consultor
10	Investigación Framework Java Spring	15 days	20/02/13 8:00	13/03/13 11:00	Consultor
11	Redación de la arquitectura principal de la aplicación	3 days	14/03/13 8:00	18/03/13 17:00	Analista
12	Modelo Entidad-Relacion de alto nivel de la Base de Datos	3 days	19/03/13 8:00	21/03/13 17:00	Analista
13	Diagrama de clases de alto nivel	2 days	22/03/13 8:00	25/03/13 17:00	Analista
14	Creación del código	30 days	26/03/13 8:00	6/05/13 17:00	Analista
15	Pruebas del código	10 days	7/05/13 7:00	20/05/13 17:00	Analista
16	Creación de la memoria del proyecto	20 days	21/05/13 7:00	17/06/13 17:00	Analista


7 ene 13
14 ene 13
21 ene 13
28 ene 13
4 feb 13
11 feb 13
18 feb 13
25 feb 13
4 mar 13
11 mar 13
18 mar 13
25 mar 13
1 Analista
2 Analista
3 Analista
4 Analista[50%];Consultor[50%]
5 Consultor
6 Consultor
7 Consultor
8 Consultor
9 Consultor
10 Consultor
11 Analista
12 Analista
13 Analista
14
15
16


1 abr 13
8 abr 13
15 abr 13
22 abr 13
29 abr 13
6 may 13
13 may 13
20 may 13
27 may 13
3 jun 13
10 jun 13
17
Analista
Analista

7.3. Marco Regulador:

La "Ley orgánica 15/1999" de Protección de Datos de Carácter Personal es una ley de obligado cumplimiento, exige el cumplimiento de una serie de artículos para proteger los datos sensibles de las personas físicas. Estos artículos, en la medida de lo posible, serán respetados en la aplicación.

Toda la información que se muestra a continuación provine directamente del BOE del estado http://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf

Artículo 1. Objeto.

La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

Artículo 2. Ámbito de aplicación

1. La presente Ley Orgánica será de aplicación a los datos de carácter personal registrados en soporte físico, que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado.

Se regirá por la presente Ley Orgánica todo tratamiento de datos de carácter personal:

a. Cuando el tratamiento sea efectuado en territorio español en el marco de las actividades de un establecimiento del responsable del tratamiento.

b. Cuando al responsable del tratamiento no establecido en territorio español, le sea de aplicación la legislación española en aplicación de normas de Derecho Internacional público.

c. Cuando el responsable del tratamiento no esté establecido en territorio de la Unión Europea y utilice en el tratamiento de datos medios situados en territorio español, salvo que tales medios se utilicen únicamente con fines de tránsito.

2. El régimen de protección de los datos de carácter personal que se establece en la presente Ley Orgánica no será de aplicación:

a. A los ficheros mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas.

b. A los ficheros sometidos a la normativa sobre protección de materias clasificadas.

c. A los ficheros establecidos para la investigación del terrorismo y de formas graves de delincuencia organizada. No obstante, en estos supuestos el responsable del fichero comunicará previamente la existencia del mismo, sus características generales y su finalidad a la Agencia de Protección de Datos.

3. Se regirán por sus disposiciones específicas, y por lo especialmente previsto, en su caso, por esta Ley Orgánica los siguientes tratamientos de datos personales:

a. Los ficheros regulados por la legislación de régimen electoral.

b. Los que sirvan a fines exclusivamente estadísticos, y estén amparados por la legislación estatal o autonómica sobre la función estadística pública.

c. Los que tengan por objeto el almacenamiento de los datos contenidos en los informes personales de calificación a que se refiere la legislación del régimen del personal de las Fuerzas Armadas.

d. Los derivados del Registro Civil y del Registro Central de penados y rebeldes.

e. Los procedentes de imágenes y sonidos obtenidos mediante la utilización de videocámaras por las Fuerzas y Cuerpos de Seguridad, de conformidad con la legislación sobre la materia.

Artículo 3. Definiciones.

A los efectos de la presente Ley Orgánica se entenderá por:

a. Datos de carácter personal: cualquier información concerniente a personas físicas identificadas o identificables.

b. Fichero: todo conjunto organizado de datos de carácter personal, cualquiera que fuere la forma o modalidad de su creación, almacenamiento, organización y acceso.

c. Tratamiento de datos: operaciones y procedimientos técnicos de carácter automatizado o no, que permitan la recogida, grabación, conservación, elaboración, modificación, bloqueo y cancelación, así como las cesiones de datos que resulten de comunicaciones, consultas, interconexiones y transferencias.

d. Responsable del fichero o tratamiento: persona física o jurídica, de naturaleza pública o privada, u órgano administrativo, que decida sobre la finalidad, contenido y uso del tratamiento.

e. Afectado o interesado: persona física titular de los datos que sean objeto del tratamiento a que se refiere el apartado c) del presente artículo.

f. Procedimiento de disociación: todo tratamiento de datos personales de modo que la información que se obtenga no pueda asociarse a persona identificada o identificable.

g. Encargado del tratamiento: la persona física o jurídica, autoridad pública, servicio o cualquier otro organismo que, sólo o conjuntamente con otros, trate datos personales por cuenta del responsable del tratamiento.

h. Consentimiento del interesado: toda manifestación de voluntad, libre, inequívoca, específica e informada, mediante la que el interesado consienta el tratamiento de datos personales que le conciernen.

i. Cesión o comunicación de datos: toda revelación de datos realizada a una persona distinta del interesado.

j. Fuentes accesibles al público: aquellos ficheros cuya consulta puede ser realizada, por cualquier persona, no impedida por una norma limitativa o sin más exigencia que, en su caso, el abono de una contraprestación. Tienen la consideración de fuentes de acceso público, exclusivamente, el censo promocional, los repertorios telefónicos en los términos previstos por su normativa específica y las listas de personas pertenecientes a grupos de profesionales que contengan únicamente los datos de nombre, título, profesión, actividad, grado académico, dirección e indicación de su pertenencia al grupo. Asimismo, tienen el carácter de fuentes de acceso público los diarios y boletines oficiales y los medios de comunicación.

8. ANEXOS

ANEXO 1: hibernate.cnf.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

       <session-factory>
              <property name="dialect">
                     org.hibernate.dialect.Oracle9Dialect
              </property>
              <property name="connection.url">
                     jdbc:oracle:thin:@localhost:1521:XE
              </property>
              <property name="connection.username">rivas</property>
              <property name="connection.password">rivas</property>
              <property name="connection.driver_class">
                     oracle.jdbc.driver.OracleDriver
              </property>
              <property name="myeclipse.connection.profile">
                     master BIBLIOTECA
              </property>
              <mapping resource="com/rivas/hibernate/Usuarios.hbm.xml" />

       </session-factory>

</hibernate-configuration>
```

ANEXO 2: clase de persistencia

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.rivas.hibernate.Usuarios" table="USUARIOS"
schema="RIVAS">
        <id name="codigoUsuario" type="java.lang.Long">
            <column name="CODIGO_USUARIO" precision="11" scale="0" />
            <generator class="assigned" />
        </id>
        <property name="nombre" type="java.lang.String">
            <column name="NOMBRE" length="80" />
        </property>
        <property name="apellidos" type="java.lang.String">
            <column name="APELLIDOS" length="140" />
        </property>
        <property name="fechaNacimiento" type="java.util.Date">
            <column name="FECHA_NACIMIENTO" length="7" />
        </property>
        <property name="email" type="java.lang.String">
            <column name="EMAIL" length="80" />
        </property>
        <property name="activo" type="java.lang.String">
            <column name="ACTIVO" length="2" />
        </property>
    </class>
</hibernate-mapping>
```

ANEXO 3: HibernateSessionFactory.java

```
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;
/**
 *  Configura las sesiones de Hibernate.
 *  Proporciona el acceso de sesiones Hibernate al hilo principal del programa
 */
public class HibernateSessionFactory {

       //INDICAMOS LA RUTA DE NUESTRO FICHERO DE CONFIGURACION DE HIBERNATE
       private static String configFile = "/com/rivas/hibernate/hibernate.cfg.xml";
       //DEFINIMOS DIFERENTES CLASES ITNERNAS DE HIBERNATE
       private    static    final    ThreadLocal<Session>    threadLocal    =    new
ThreadLocal<Session>();
    private static Configuration configuration = new Configuration();
    private static org.hibernate.SessionFactory sessionFactory;

       static {
       try {
              //CARGAMOS NUESTRO FICHERO DE CONFIGURACION Y CREAMOS LA SESION
                     configuration.configure(configFile);
                     sessionFactory = configuration.buildSessionFactory();
              } catch (Exception e) {
                     System.err.println("%%%% Error Creating SessionFactory %%%%");
                     e.printStackTrace();
              }
    }
       /**
     * Devuelve la instancia de la Sesion Hibernate.
     * El código usará el <code>SessionFactory</code> anteriormente creado.
     *  @return Session
     *  @throws HibernateException
     */
    public static Session getSession() throws HibernateException {
        Session session = (Session) threadLocal.get();

              if (session == null || !session.isOpen()) {
                     //Recargamos el sessionFactory si fuese necesario
                     if (sessionFactory == null) {
                            rebuildSessionFactory();
                     }
                     session      =      (sessionFactory      !=      null)      ?
sessionFactory.openSession():null;
                     threadLocal.set(session);
              }
         return session;
    }
       /**
     *  Recargamos el Hibernate Session Factory
     */
       public static void rebuildSessionFactory() {
              try {
                     configuration.configure(configFile);
                     sessionFactory = configuration.buildSessionFactory();
              } catch (Exception e) {
                     System.err.println("%%%% Error Creating SessionFactory %%%%");
                     e.printStackTrace();
              }
       }
}
```

ANEXO 4: AplicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans default-lazy-init="true"
	default-autowire="byName"
xmlns="http://www.springframework.org/schema/beans"
	xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
	http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.1.xsd">

	<bean id="prueba"
	class="com.rivas.controlador.Prueba" scope="prototype" />


	<!-- >>>>>>>>>>>>>> ZONA DE APLICACION <<<<<<<<<<<<<<<< -->
	<!-- SINGLENTON QUE RESUELVE EL ACCESO EL APPLICATIONCONTEXT EN EL
PROGRAMA -->
	<bean                                                id="acceso_contextos"
class="com.rivas.spring.Acceso_ApplicationContext"
		scope="singleton" lazy-init="false" />

</beans>
```

ANEXO 5: Acceso_AplicationContext.java

```
package com.rivas.spring;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

public class Acceso_ApplicationContext implements ApplicationContextAware {

	private static ApplicationContext ctx;

	/**
	 * Recibimos de forma automatica (al crearse el applicationcontext)
cuando
	 * se crea el bean.<br/>
	 */
	@Override
	public void setApplicationContext(ApplicationContext ctx)
			throws BeansException {
		Acceso_ApplicationContext.ctx = ctx;
	}

	/**
	 * Nos permite acceder al WebapplicationContext directamente en
cualquier clase de aplicacion.
	 * @return
	 */
	public static ApplicationContext getContextoSpring() {
		return ctx;
	}

	/**
	 * Proceso para la obtencion directa del bean.
	 *
	 * @param objeto
	 *            Id o nombre del objeto que spring tiene que pasar.
	 * @return Objeto creado por spring.
	 */
	public static Object getBean(String objeto) {
		return ctx.getBean(objeto);

	}
}
```

ANEXO 6: Código SQL, creación de entidades y relaciones

Al tener relaciones entre tablas el orden de creación de las mismas SI es importante, por lo que no podremos crear una entidad con una FK a otra que no exista. El código de creación de la Base de datos el siguiente:

```
/* Creamos Todos los sequence que necesitamos para nuestras entidades */

CREATE SEQUENCE   "CODIGO_EDIT"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_ESCRITOR"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_GENERO"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_IDIOMA"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_LIBROS"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_PRESTAMOS"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1

CREATE SEQUENCE   "CODIGO_USUARIO"  MINVALUE 1 MAXVALUE
9999999999999999999999999 INCREMENT BY 1


/* Creamos entidad País */

CREATE TABLE  "PAIS"
   (   "CODIGO_PAIS" NUMBER(11,0) NOT NULL ENABLE,
       "PAIS_ISONUM" NUMBER(6,0),
       "PAIS_ISO2" CHAR(2),
       "PAIS_ISO3" CHAR(3),
       "PAIS_NOMBRE" VARCHAR2(80),
        CONSTRAINT "PAIS_CODIGO_PAIS_PK" PRIMARY KEY ("CODIGO_PAIS") ENABLE
   )


/* Creamos entidad Genero */

CREATE TABLE  "GENERO"
   (   "CODIGO_GENERO" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE_GENERO" VARCHAR2(60),
        CONSTRAINT  "GENERO_CODIGO_GENERO_PK"  PRIMARY  KEY  ("CODIGO_GENERO")
ENABLE
   )
```

```
/* Creamos entidad Idioma */

CREATE TABLE  "IDIOMA"
   (   "CODIGO_IDIOMA" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE_IDIOMA" VARCHAR2(60),
        CONSTRAINT  "IDIOMA_CODIGO_IDIOMA_PK"  PRIMARY  KEY  ("CODIGO_IDIOMA")
ENABLE
   )


/* Creamos entidad Editoriales */

CREATE TABLE  "EDITORIALES"
   (   "CODIGO_EDIT" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE_EDITORIAL" VARCHAR2(80),
       "PAGINA_WEB" VARCHAR2(200),
       "CODIGO_PAIS_ED" NUMBER(11,0),
        CONSTRAINT  "EDITORIALES_CODIGO_EDIT_PK"  PRIMARY  KEY  ("CODIGO_EDIT")
ENABLE,
        CONSTRAINT "CODIGO_PAIS_ED_FK" FOREIGN KEY ("CODIGO_PAIS_ED")
         REFERENCES  "PAIS" ("CODIGO_PAIS") ENABLE
   )


/* Creamos entidad Escritores y sus relaciones */

CREATE TABLE  "ESCRITORES"
   (   "CODIGO_ESCRITOR" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE" VARCHAR2(80),
       "APELLIDOS" VARCHAR2(140),
       "FECHA_NACIMIENTO" DATE,
       "CODIGO_PAIS" NUMBER(11,0),
       "ELIMINADO" VARCHAR2(3),
        CONSTRAINT      "ESCRITORES_CODIGO_ESCRITOR_PK"      PRIMARY      KEY
("CODIGO_ESCRITOR") ENABLE,
        CONSTRAINT "CODIGO_PAIS_FK" FOREIGN KEY ("CODIGO_PAIS")
         REFERENCES  "PAIS" ("CODIGO_PAIS") ENABLE
   )


/* Creamos entidad Libros y sus relaciones */

CREATE TABLE  "LIBROS"
   (   "CODIGO_LIBROS" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE_LIBRO" VARCHAR2(80),
       "FECHA_LANZAMIENTO" DATE,
       "INFO" VARCHAR2(200),
       "CODIGO_GENERO_LI" NUMBER(11,0),
       "CODIGO_EDIT_LI" NUMBER(11,0),
       "CODIGO_ESCRITOR_LI" NUMBER(11,0),
       "CODIGO_IDIOMA_LI" NUMBER(11,0),
       "CANTIDAD" NUMBER(11,0),
       "DISPONIBLE" VARCHAR2(3),
       "ELIMINADO" VARCHAR2(3),
       "PORTADA" BLOB,
        CONSTRAINT  "LIBROS_CODIGO_LIBROS_PK"  PRIMARY  KEY  ("CODIGO_LIBROS")
ENABLE,
        CONSTRAINT "CODIGO_GENERO_LI_FK" FOREIGN KEY ("CODIGO_GENERO_LI")
```

```
        REFERENCES  "GENERO" ("CODIGO_GENERO") ENABLE,
       CONSTRAINT "CODIGO_EDIT_LI_FK" FOREIGN KEY ("CODIGO_EDIT_LI")
        REFERENCES  "EDITORIALES" ("CODIGO_EDIT") ENABLE,
       CONSTRAINT "CODIGO_ESCRITOR_LI_FK" FOREIGN KEY ("CODIGO_ESCRITOR_LI")
        REFERENCES  "ESCRITORES" ("CODIGO_ESCRITOR") ENABLE,
       CONSTRAINT "CODIGO_IDIOMA_LI_FK" FOREIGN KEY ("CODIGO_IDIOMA_LI")
        REFERENCES  "IDIOMA" ("CODIGO_IDIOMA") ENABLE
   )
```

```
/* Creamos entidad Usuarios y sus relaciones */

CREATE TABLE  "USUARIOS"
   (   "CODIGO_USUARIO" NUMBER(11,0) NOT NULL ENABLE,
       "NOMBRE" VARCHAR2(80),
       "APELLIDOS" VARCHAR2(140),
       "EMAIL" VARCHAR2(80),
       "MOROSO" VARCHAR2(3),
       "ELIMINADO" VARCHAR2(3),
       "NUMERO_LIBROS_PRESTADOS" NUMBER(11,0),
        CONSTRAINT       "USUARIOS_CODIGO_USUARIO_PK"       PRIMARY       KEY
("CODIGO_USUARIO") ENABLE
   )
```

```
/* Creamos entidad Prestamos y sus relaciones */

CREATE TABLE  "PRESTAMOS"
   (   "CODIGO_PRESTAMOS" NUMBER(11,0) NOT NULL ENABLE,
       "FECHA_PRESTAMO" DATE,
       "DEVUELTO" VARCHAR2(3),
       "CODIGO_LIBROS_PRE" NUMBER(11,0),
       "CODIGO_USU_PRE" NUMBER(11,0),
        CONSTRAINT      "PRESTAMOS_CODIGO_PRESTAMOS_PK"      PRIMARY      KEY
("CODIGO_PRESTAMOS") ENABLE,
       CONSTRAINT "CODIGO_LIBROS_PRE_FK" FOREIGN KEY ("CODIGO_LIBROS_PRE")
        REFERENCES  "LIBROS" ("CODIGO_LIBROS") ENABLE,
       CONSTRAINT "CODIGO_USU_PRE_FK" FOREIGN KEY ("CODIGO_USU_PRE")
        REFERENCES  "USUARIOS" ("CODIGO_USUARIO") ENABLE
   )
```

```
/* Una vez creados los sequence y las entidades asignamos los sequence a las
PK de las entidades mediante index.*/

CREATE UNIQUE INDEX  "EDITORIALES_CODIGO_EDIT_PK" ON  "EDITORIALES"
("CODIGO_EDIT")

CREATE UNIQUE INDEX  "ESCRITORES_CODIGO_ESCRITOR_PK" ON  "ESCRITORES"
("CODIGO_ESCRITOR")

CREATE UNIQUE INDEX  "GENERO_CODIGO_GENERO_PK" ON  "GENERO" ("CODIGO_GENERO")

CREATE UNIQUE INDEX  "IDIOMA_CODIGO_IDIOMA_PK" ON  "IDIOMA" ("CODIGO_IDIOMA")

CREATE UNIQUE INDEX  "LIBROS_CODIGO_LIBROS_PK" ON  "LIBROS" ("CODIGO_LIBROS")

CREATE UNIQUE INDEX  "PAIS_CODIGO_PAIS_PK" ON  "PAIS" ("CODIGO_PAIS")
```

```
CREATE UNIQUE INDEX  "PRESTAMOS_CODIGO_PRESTAMOS_PK" ON  "PRESTAMOS"
("CODIGO_PRESTAMOS")

CREATE UNIQUE INDEX  "USUARIOS_CODIGO_USUARIO_PK" ON  "USUARIOS"
("CODIGO_USUARIO");

/* Es muy recomendable crear los índices a la vez que las tablas para no
omitir ninguno de ellos. Pero al exportar la Base de Datos se nos mostró de
esta forma. */
```

9. BIBLIOGRAFÍA

9.1. Libros

- Richardson, Avondolio, Schrager, Mitchell, Scanlon, 2007. Java JDK 6. Anaya Multimedia (fecha de consulta Marzo 2013).
- Rogers Cadenhead, 2012. Java 7. Anaya Multimedia (fecha de consulta Febrero 2013).
- Christian Bauer, Gavin King. 2008. Hibernate in Action. Manning (fecha de consulta Marzo 2013).
- Emanuel Bernard, John Griffin. 2009. Hibernate Search in Action. Manning (fecha de consulta Marzo 2013).
- Christian Bauer, Gavin King. 2008. Java Persistence with Hibernate. Manning (fecha de consulta Marzo 2013).
- Craig Walls, Ryan Breidenbach. 2008. Spring in Action. Manning (fecha de consulta Abril 2013).
- Craig Walls. 2011. Spring in Action 3ª edición. Manning (fecha de consulta Abril 2013).
- Matthew Robinson, Pavel Vorobiev. 2003. Swing 2ª edición. Manning (fecha de consulta Abril 2013).
-

9.2. Direcciones Web.

- www.javahispano.org (fecha de consulta Mayo 2013)
- www.hibernate.org (fecha de consulta Junio 2013)
- www.springsource.org (fecha de consulta Junio 2013)
- http://rekkeb.wordpress.com/2012/05/13/por-que-spring-simplifica-el-desarrollo-de-nuestras-aplicaciones-java/ (fecha de consulta Junio 2013)
- http://www.genbetadev.com/java-j2ee/spring-framework-introduccion (fecha de consulta Junio 2013)

- http://jcesarperez.blogspot.com.es/2008/09/cuando-usar-no-usar-hibernatede.html (fecha de consulta Julio 2013)
- http://swing-facil.blogspot.com.es/2011/04/introduccion.html (fecha de consulta Julio 2013)
- http://diccionario.babylon.com/java/ (fecha de consulta Julio 2013)
- http://www.revista.unam.mx/vol.1/num2/art4/ (fecha de consulta Julio 2013)
- http://www.cad.com.mx/historia_del_lenguaje_java.htm (fecha de consulta Julio 2013)
- http://zarza.usal.es/~fgarcia/doc/tuto2/I_2.htm (fecha de consulta Julio 2013)

- http://www.slideshare.net/ZulmaBautista/java-25177397 (fecha de consulta Julio 2013)
- http://www.ecured.cu/index.php/Hibernate (fecha de consulta Agosto 2013)
- http://www.cursohibernate.es/doku.php?id=unidades:01_introduccion_orm:06_intro_orm (fecha de consulta Agosto 2013)
- http://osl2.uca.es/wikiCE/index.php/Hibernate (fecha de consulta Agosto 2013)
- http://www.elclubdelprogramador.com/2012/01/09/spring-introduccion-a-spring/ (fecha de consulta Agosto 2013)
- http://www.sicuma.uma.es/sicuma/independientes/argentina08/Badaracco/spprincar.htm (fecha de consulta Agosto 2013)
- http://es.wikipedia.org/wiki/Spring_Framework (fecha de consulta Agosto 2013)
- http://www.slideshare.net/neodevelop/spring-presentation (fecha de consulta Agosto 2013)
- http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf (fecha de consulta Agosto 2013)
- http://pablolg.wikispaces.com/file/view/spring_tutorial_v0.271.pdf (fecha de consulta Agosto 2013)