

# LICENCIA GPL

---

Puede copiar y distribuir el Programa (o un trabajo basado en él, según se especifica en el apartado 2, como código objeto o en formato ejecutable según los términos de los apartados 1 y 2, suponiendo que además cumpla una de las siguientes condiciones:

1. Acompañarlo con el código fuente completo correspondiente, en formato electrónico, que debe ser distribuido según se especifica en los apartados 1 y 2 de esta Licencia en un medio habitualmente utilizado para el intercambio de programas, o
2. Acompañarlo con una oferta por escrito, válida durante al menos tres años, de proporcionar a cualquier tercera parte una copia completa en formato electrónico del código fuente correspondiente, a un coste no mayor que el de realizar físicamente la distribución del fuente, que será distribuido bajo las condiciones descritas en los apartados 1 y 2 anteriores, en un medio habitualmente utilizado para el intercambio de programas, o
3. Acompañarlo con la información que recibió ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribución no comercial y sólo si usted recibió el programa como código objeto o en formato ejecutable con tal oferta, de acuerdo con el apartado 2 anterior).

# PFC: J2EE

Diseño e implementación de un framework de persistencia

16/01/2012

Ingeniería Informática UOC

José M<sup>a</sup> Casablanca González

Consultor: Oscar Escudero Sánchez



## INDICE

Índice de Ilustraciones .....	6
Índice de Tablas.....	7
Resumen del proyecto .....	8
Introducción .....	9
1.1. Descripción del PFC.....	9
1.2. Objetivos generales y específicos .....	10
1.3. Planificación .....	10
1.4. Diagrama de GANTT .....	11
2. Características y definición de un framework .....	13
2.1. ¿Qué es un framework?.....	13
2.2. El patrón MVC .....	14
2.3. Arquitectura J2EE .....	15
3. Persistencia.....	17
3.1. Persistencia de objetos .....	17
3.1.1. Acceso directo a la base de datos .....	17
3.1.2. Mapeadores.....	18
3.1.3. Generadores de código .....	18
3.1.4. Orientación a aspectos .....	19
3.1.5. Lenguajes orientados a objetos.....	19
3.1.6. Esquemas de prevalencia .....	20
3.2. Frameworks de persistencia .....	20
3.2.1. IBATIS.....	20
3.2.2. HIBERNATE .....	20
3.2.3. EJB (Enterprise Java Beans).....	21
3.2.4. JDO (Java Data Object).....	22
3.2.5. Comparación de los diferentes frameworks de persistencia .....	23
4. Diseño de un framework de persistencia .....	25
4.1. Introducción.....	25
4.2. Características .....	25
4.3. Mapeo.....	25
4.4. Análisis y Diseño del framework .....	26
4.4.1. Conexión y relación de la Base de Datos .....	26

4.4.2.	<i>Sistema de excepciones</i>	28
4.4.3.	<i>Diagrama de paquetes</i>	28
4.4.4.	<i>Diagrama de secuencias</i>	30
4.4.4.1.	<i>Secuencia de creación</i>	31
4.4.4.2.	<i>Secuencia de eliminación</i>	32
4.4.4.3.	<i>Secuencia de consulta</i>	33
4.4.4.4.	<i>Secuencia de modificación</i>	34
4.4.5.	<i>Diagrama de clases</i>	35
5.	<i>Aplicación de ejemplo</i>	36
5.1.	<i>Introducción</i>	36
5.2.	<i>Uso de la aplicación</i>	37
5.2.1.	<i>Listado de clientes</i>	37
5.2.2.	<i>Añadir un nuevo cliente</i>	39
5.2.3.	<i>Modificar un cliente ya existente</i>	41
5.2.4.	<i>Eliminar un cliente</i>	42
5.3.	<i>Instalación de la aplicación</i>	43
5.3.1.	<i>Java Runtime Environment (JRE)</i>	43
5.3.2.	<i>Tomcat</i>	43
5.3.3.	<i>MySQL</i>	44
5.3.4.	<i>Instalación de la aplicación</i>	44
6.	<i>Conclusiones</i>	46
6.1.	<i>Objetivos cumplidos</i>	46
6.2.	<i>Ampliaciones del framework</i>	46
7.	<i>Glosario</i>	48
8.	<i>Bibliografía</i>	50
8.1.	<i>Libros</i>	50
8.2.	<i>Direcciones web</i>	50
I.	<i>Anexo</i>	51

## Índice de Ilustraciones

Ilustración 1.- Diagrama de Gantt .....	12
Ilustración 2.- El patrón MVC .....	14
Ilustración 3.- Arquitectura J2EE .....	16
Ilustración 4.- Esquema BMP .....	22
Ilustración 5.- Esquema CMP .....	22
Ilustración 6.- Gráfica opciones de trabajo J2EE .....	24
Ilustración 7.- Gráfica cantidad de soporte J2EE .....	24
Ilustración 8.- Clase DataSourceBasicoImpl .....	27
Ilustración 9.- Clase FWPersistenciaExceptionImpl .....	28
Ilustración 10.- Diagrama de paquetes.....	29
Ilustración 11.- Diagrama de secuencias .....	30
Ilustración 12.- Secuencia de creación .....	31
Ilustración 13.- Secuencia de eliminación .....	32
Ilustración 14.- Secuencia de consulta .....	33
Ilustración 15.- Secuencia de modificación.....	34
Ilustración 16.- Diagrama de clases .....	35
Ilustración 17.- Modelo de datos aplicación de ejemplo.....	36
Ilustración 18.- Pantalla de listado de la aplicación de ejemplo .....	38
Ilustración 19.- Pantalla de creación de cliente .....	39
Ilustración 20.- Listado de clientes (Creación) .....	40
Ilustración 22.- Formulario de modificación.....	41
Ilustración 21.- Listado previo a la modificación de un cliente.....	41
Ilustración 23.- Listado tras la modificación de un cliente.....	42
Ilustración 24.- Listado previo a la eliminación de un cliente .....	42
Ilustración 25.- Listado de clientes tras la eliminación .....	43
Ilustración 26.- Pantalla inicial de la aplicación .....	45

## Índice de Tablas

Tabla 1.- Planificación del desarrollo del proyecto .....	11
Tabla 2.- Ventajas del uso de un framework .....	13
Tabla 3.- Desventajas del uso de un framework .....	14

## Resumen del proyecto

El principal objetivo de este proyecto es el análisis y desarrollo de un conjunto de componentes que simplifiquen y agilicen el desarrollo de la capa de persistencia en aplicaciones desarrolladas especialmente con J2EE o que admitan la interacción con esta tecnología de desarrollo a medida.

Una de las partes más complejas que hay que resolver al plantear cualquier aplicación es el acceso a los datos persistentes que tiene que gestionar. En aplicaciones J2EE hay diversas opciones para implementar este acceso: uso de JDBC directamente, uso de EJBs, uso de marcos de trabajo que faciliten un mapeo entidad-relación, etc. En concreto lo que se abordará en el desarrollo de este proyecto es la creación de un marco de trabajo que nos facilite esta función.

Lo primero que será hacer un estudio de los conceptos y términos que se emplean en la capa de persistencia para tener un conocimiento de la misma. Una vez que tengamos la suficiente información de lo que vamos a tener que realizar, evaluaremos las alternativas existentes, en concreto como hemos dicho anteriormente en el marco de trabajo.

Una vez que tengamos analizado los marcos de trabajo más importantes actualmente, se realizará el diseño y la implementación de un framework (marco de trabajo) que mediante J2EE nos permita cubrir las necesidades de este y las funcionalidades básicas.

Este framework está diseñado con un conjunto de clases que garantizan la abstracción para el usuario de la conexión y comunicación con la base de datos relacional.

Por último se va a implementar una aplicación de ejemplo que permita mostrar las funcionalidades implementadas con un grado de abstracción para el usuario muy alto.

## Introducción

### 1.1. Descripción del PFC

La principal motivación para la realización de este PFC es que a nivel empresarial absolutamente todo se encuentra almacenado en datos, desde facturaciones hasta los datos de los empleados.

Los frameworks son unas herramientas básicas en el desarrollo ágil de aplicaciones, ya que se basan en la reutilización de unas funcionalidades que ya se encuentran implementadas. La implementación de este PFC no busca algo diferente de esto, sino algo que lo complemente.

El uso de este Framework nos facilitará el acceso a una base de datos relacional, para poder gestionar la información almacenada en ella. La utilización de este Framework nos permitirá independizar la aplicación que muestre los datos de la parte que se encargue de su gestión.

El objetivo principal de este proyecto es el de indagar en la arquitectura interna de los frameworks de persistencia. La capa de persistencia dentro de un proyecto que cumple el patrón MVC (Modelo – Vista – Controlador) es una de las partes más importantes, puesto que nos permite gestionar y almacenar información para su uso posterior.

A su vez otro objetivo importante es el de realizar un conjunto de clases que permitan exportarlo de forma que para el futuro usuario sea totalmente transparente el acceso a un sistema de datos normalmente gestionado por bases de datos relacionales. El lenguaje con el que se realizará el Framework será JAVA, lenguaje de contrastada importancia basado en OO (Orientación a Objetos). Esta filosofía nos permitirá crear una Framework extensible y escalable.

Por último se implementará una pequeña aplicación que haga un uso correcto del framework implementado. Para la implementación de esta aplicación, se utilizará la tecnología JSF (Java Server Face), tecnología que como su nombre indica se integra perfectamente con JAVA.

## 1.2. *Objetivos generales y específicos*

El desarrollo del proyecto final de carrera tiene como finalidad principal la implementación de un Framework de persistencia. Los objetivos específicos del proyecto son:

- Definición de Framework: características y tipos de Framework existentes.
- Persistencia de datos: ¿Qué es?, características y Frameworks específicos.
- Frameworks de persistencia: Búsqueda de los Frameworks de persistencia que existen actualmente. Analizar su funcionamiento y compararlos comprobando sus ventajas e inconvenientes.
- Diseño de un Framework de persistencia: Valiéndonos de los Frameworks estudiados anteriormente, diseñar e implementar un nuevo Framework que basándose en los anteriores, nos permita el desarrollo de una nueva funcionalidad. Se debe desarrollar con tecnología Java.
- Desarrollo de una aplicación que haga uso del Framework desarrollado y demuestre su buen funcionamiento.

## 1.3. *Planificación*

Para la realización de la planificación, se han tenido en cuenta las fechas de entrega propuestas por los consultores de la asignatura. La planificación que se debe cumplir en el desarrollo del proyecto final de carrera, debe ser la siguiente:

ID	Tarea	Duración	F. Inicio	F. Fin	Predecesor
0.1	Plan de trabajo	10d	23/09/2011	06/10/2011	
1	Entrega PEC1	1d	07/10/2011	07/10/2011	0.1
1.1	Datos Generales Frameworks	5d	10/10/2011	14/10/2011	
1.2	Análisis de la persistencia de datos	6d	17/10/2011	24/10/2011	1.1
1.3	Evaluación de frameworks de	12d	25/10/2011	09/11/2011	1.2

	persistencia				
2	Entrega PEC2	1d	10/11/2011	10/11/2011	1.3
2.1	Análisis y diseño del framework	10d	11/11/2011	24/11/2011	
2.2	Implementación del framework	16d	25/11/2011	16/12/2011	2.1
3	Entrega PEC3	1d	19/12/2011	19/12/2011	2.2
3.1	Aplicación de ejemplo y pruebas	10d	20/12/2011	02/01/2012	
3.2	Memoria	6d	03/01/2012	10/01/2012	3.1
3.3	Elaboración presentación	3d	11/01/2012	13/01/2012	3.2
4	Entrega final	1d	16/01/2012	16/01/2012	3.3

**Tabla 1.- Planificación del desarrollo del proyecto**

#### *1.4. Diagrama de GANTT*

Mediante este diagrama de Gantt vemos de forma más gráfica lo mismo que se ha expuesto en el apartado 1.3 de este documento. En esta gráfica podemos ver como se separan las diferentes fases por las que ha pasado el proyecto.

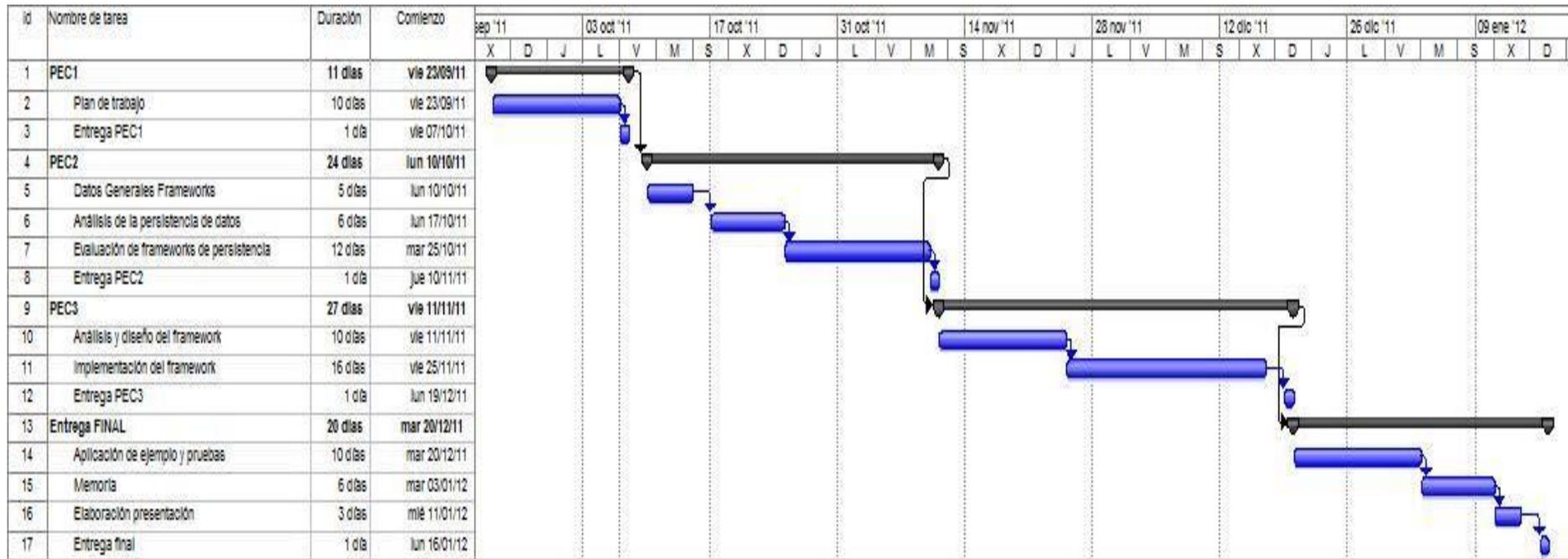


Ilustración 1.- Diagrama de Gantt

## 2. Características y definición de un framework

### 2.1. ¿Qué es un framework?

Un framework representa una Arquitectura de Software que modela las relaciones que existen entre las entidades del dominio y al mismo tiempo que proporciona una metodología y una estructura de trabajo que se puede extender a las aplicaciones del dominio que en algunos casos puedan utilizarlos.

Básicamente, los frameworks son construidos en base a lenguajes orientados a objetos. Esto permite la modularidad de los componentes y una óptima reutilización de código. Además, en la mayoría de los casos, cada framework específico implementará uno o más patrones de diseño de software que aseguren la escalabilidad del producto.

Un framework de aplicaciones es un término utilizado para referirse a un conjunto de bibliotecas o clases utilizadas para implementar la estructura estándar de una aplicación. En otras palabras, es un esquema o patrón para el desarrollo o implementación de una aplicación completa, o de una parte específica de una aplicación.

Las ventajas y desventajas de la utilización de un framework se resumen en la siguiente tabla:

VENTAJAS	
<b>Infraestructura prefabricada</b>	Presentan clases ya creadas y que funcionan, de forma que simplifican la labor del usuario
<b>Proporción de una arquitectura</b>	Los frameworks no muestran su funcionamiento interno, de forma que para el usuario es mucho más transparente su utilización
<b>Reducción del mantenimiento</b>	Al ser unos componentes cerrados, no es necesario el mantenimiento del framework por parte de la persona que lo esté usando.
<b>Base industria de componentes</b>	Un framework bien diseñado puede ser la base de un sinnúmero de componentes.

Tabla 2.- Ventajas del uso de un framework

DESVENTAJAS	
<b>Limitación de flexibilidad</b>	Al ser un producto cerrado, todo lo que el usuario construya para adaptarse a un framework, deberá cumplir con sus exigencias de programación.
<b>Dificultad de aprendizaje</b>	Todo framework necesita un tiempo de aprendizaje por parte del usuario para poder utilizarlo de una forma más útil.
<b>Reducción de creatividad</b>	El framework obliga a que los usuario se adapten a sus criterios de desarrollo.

Tabla 3.- Desventajas del uso de un framework

## 2.2. El patrón MVC

El patrón MVC (Modelo – Vista - Controlador) es una guía para el diseño de aplicaciones que comprendan de las siguientes capas:

- Modelo, representa los datos de la aplicación y contiene la lógica de negocio de la aplicación en concreto. Los datos de la aplicación, suelen estar en bases de datos relacionales.
- Vista, capa visual que permite la iteración del usuario con la aplicación. Permite introducir y obtener datos del modelo.
- Controlador, conjunto que procesa las peticiones desde la capa de la vista y las asigna a su modelo correspondiente.

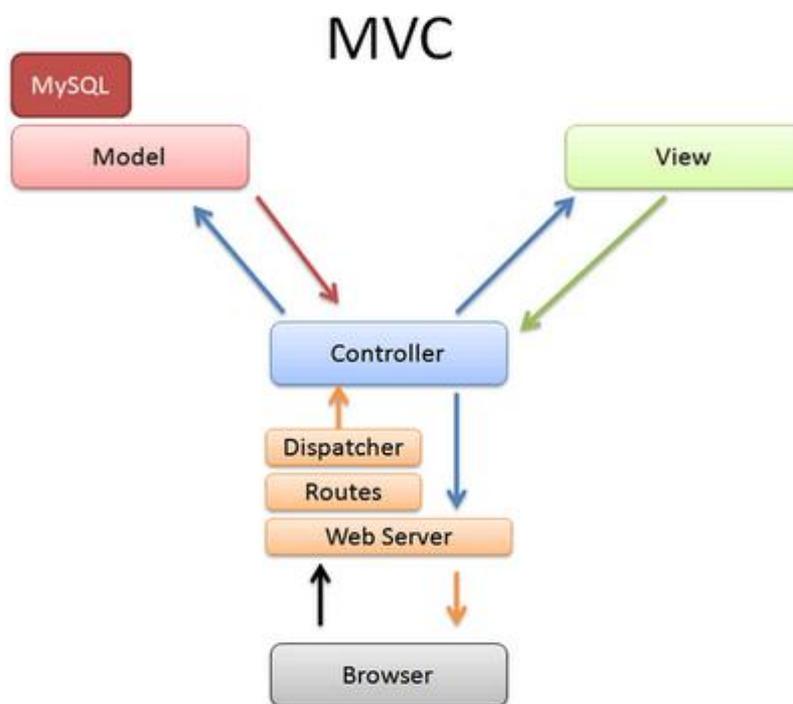


Ilustración 2.- El patrón MVC

MVC nos aporta una construcción de software muy mantenible, en la que se pueden localizar de forma ágil los errores. Supone un diseño modular, y muy poco acoplado, favoreciendo la reutilización.

Por ejemplo, podemos realizar una interfaz gráfica de escritorio, y una web, que compartirían las capas Controlador y Modelo, y solo trataríamos como desarrollos distintos las dos capas Vista.

### 2.3. *Arquitectura J2EE*

La arquitectura J2EE es muy parecida a la arquitectura que comprende el patrón del Modelo – Vista – Controlador, con la diferencia de que es más extendido ya que comprende 4 capas en lugar de 3. Estas capas son las siguientes:

- Capa de **cliente**, también conocida como capa de presentación o de aplicación. Nos encontramos con componentes Java (applets o aplicaciones) y no-Java (HTML, JavaScript, etc.).
- Capa **Web**. Intermediario entre el cliente y otras capas. Sus componentes principales son los servlets y las JSP. Aunque componentes de capa cliente (applets o aplicaciones) pueden acceder directamente a la capa EJB, lo normal es que Los servlets/JSPs pueden llamar a los EJB.
- Capa **Enterprise JavaBeans**. Permite a múltiples aplicaciones tener acceso de forma concurrente a datos y lógica de negocio. Los EJB se encuentran en un servidor EJB, que no es más que un **servidor de objetos distribuidos**. Un EJB puede conectarse a cualquier capa, aunque su misión esencial es conectarse con los sistemas de información empresarial (un gestor de base de datos, ERP, etc.)
- Capa de **sistemas de información empresarial**.

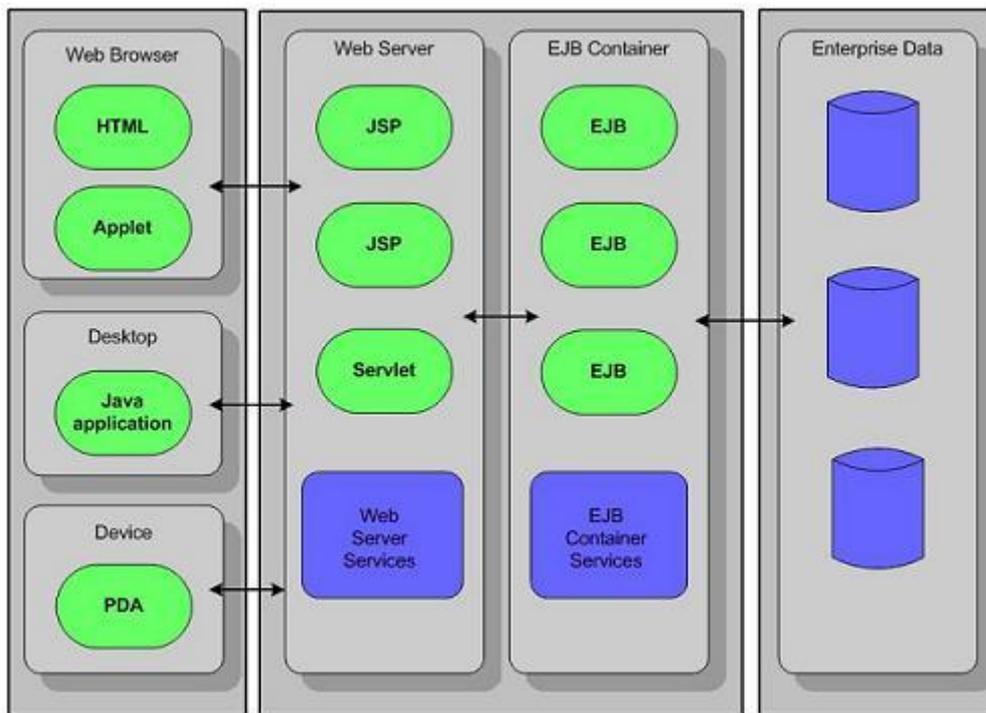


Ilustración 3.- Arquitectura J2EE

## 3. Persistencia

### 3.1. *Persistencia de objetos*

Todo procesamiento de datos requerido por una aplicación, puede ser dividido a grandes rasgos en: datos persistentes y el procesamiento de estos. Los mecanismos utilizados para acceder e interactuar con los datos son muy importantes no solo por su impacto en el desempeño final del sistema sino también desde el punto de vista de aspectos tales como mantenibilidad, diseño, reusabilidad y escalabilidad.

Existen numerosas alternativas a la hora de seleccionar un mecanismo que permita resolver la persistencia de los objetos, de forma que permita acceder a los datos y posteriormente manipularlos. Estos son los “mecanismos de persistencia” más destacados:

#### 3.1.1. *Acceso directo a la base de datos*

El mecanismo en cuestión implica el uso directo de la base de datos para implementar la persistencia del sistema. Esto último refiere al acceso a los datos utilizando por ejemplo una interfaz estandarizada en conjunto con algún lenguaje de consulta soportado directamente por el SGBD. En este caso, no existe ningún tipo de capa entre la aplicación y el SGBD utilizado.

- **Acceso Directo a Base de Datos Relacional.** Utilizando una base de datos relacional como dispositivo de almacenamiento, es la intención fundamental proveer un mecanismo de acceso a datos que maneje el modelo relacional.
- **Acceso Directo a Base de Datos Objeto – Relacional.** Utilizando una base de datos objeto-relacional como dispositivo de almacenamiento, es la intención fundamental proveer un mecanismo de acceso a datos que brinde un modelo de datos más rico que el relacional.
- **Acceso Directo a Base de Datos Orientada a Objetos.** Con una base de datos orientada a objetos como dispositivos de almacenamiento, es la intención fundamental proveer un mecanismo de acceso a datos que no implique una diferencia de paradigmas entre el nivel lógico y de persistencia.

- **Acceso Directo a Base de Datos XML.** Haciendo uso de una base de datos XML como dispositivo de almacenamiento, proveerá un mecanismo de persistencia que permita almacenar los datos de los objetos en documentos XML.

### 3.1.2. *Mapeadores*

En esta categoría se incluyen aquellos mecanismos que se basan en la traducción bidireccional entre los datos encapsulados en los objetos de la lógica de un sistema orientado a objetos y una fuente de datos que maneja un paradigma distinto. El mapeador ha de lidiar con los diferentes problemas que se presentan al mapear los datos entre paradigmas.

- **Mapeadores Objeto-Relacional.** El objetivo de este mecanismo radica en contar con un mecanismo para mapear los objetos de la lógica de un sistema orientado a objetos a una base de datos relacional.
- **Mapeadores Objeto – XML.** El cometido de este mecanismo es permitir el almacenamiento de los datos contenidos en los objetos de la lógica en forma de documentos XML. El mapeo objeto-XML provee un medio por el cual los datos de negocio pueden ser visualizados en su forma persistida, al mismo tiempo que se facilita el intercambio de dichos datos con otros sistemas.

### 3.1.3. *Generadores de código*

En el desarrollo de aplicaciones empresariales existen tareas repetitivas que el programador se ve obligado a realizar. En este caso, se pueden usar herramientas que generen código correcto y basado en patrones para resolver la persistencia del sistema, permitiendo al desarrollador centrarse en el código que resuelve la lógica de negocio.

### 3.1.4. *Orientación a aspectos*

La programación Orientada a Aspectos (AOP) es un paradigma que permite definir abstracciones que encapsulen característica que involucren a un grupo de componentes funcionales, es decir, que corten transversalmente al sistema. En la programación orientada a aspectos, las clases son diseñadas e implementadas de forma separada a los aspectos requiriendo luego una fusión. Es intención de este mecanismo manipular concretamente la persistencia como un aspecto ortogonal a las funcionalidades, desacoplando el código correspondiente al resto del sistema. Otro objetivo de este mecanismo es a su vez permitir modularizar la persistencia para luego poder reutilizar el código generado.

### 3.1.5. *Lenguajes orientados a objetos*

Se trata de resolver la persistencia de datos utilizando funcionalidades provistas por el propio lenguaje de programación evitando la inclusión de frameworks u otras herramientas ajenas al lenguaje y persistiendo los datos usando ficheros de texto plano o ficheros XML.

Al no hacer uso de frameworks adicionales, los principales mecanismos de los que se vale este mecanismo de persistencias son:

- **Librerías estándares.** Intenta utilizar las funcionalidades aportadas por las librerías propias del lenguaje de programación orientado a objetos para intentar resolver la persistencia de los datos del sistema.
- **Lenguajes persistentes.** El objetivo de este mecanismo implica resolver de manera transparente la persistencia de datos haciendo uso de funcionalidades provistas por el lenguaje de programación.
- **Lenguajes de consulta integrado.** Existen múltiples interfaces de acceso a datos que son utilizados desde numerosos lenguajes de programación orientados a objetos como métodos para administrar y consultar los datos persistentes de un sistema.

### 3.1.6. *Esquemas de prevalencia*

Son marcos de trabajo que mantienen los objetos en memoria volátil y periódicamente utilizan esquemas de serialización de objetos para guardar una imagen de los objetos de la aplicación. Es un acercamiento muy simple y rápido a la persistencia aunque presenta algunos problemas de escalabilidad.

## 3.2. *Frameworks de persistencia*

### 3.2.1. *IBATIS*

IBATIS es un framework con tecnología open-source que basado en JDBC proporciona un medio sencillo y flexible para poder mover datos entre un objeto java y una base de datos relacional. De esta forma se reduce considerablemente la cantidad de código que hay que producir para establecer una conexión con una base de datos relacional.

Este framework permite relacionar un bean de java con una sentencia SQL mediante la configuración de archivos xml. En estos ficheros xml, se crean consultas complejas que son adaptadas al sistema de gestión de base de datos que estemos utilizando en el proyecto. Esto supone un gran inconveniente, ya que IBATIS no es independiente al proveedor de Base de Datos, de forma que si se cambiase este, habría que editar todas las sentencias definidas en los ficheros xml.

Aparte de ese inconveniente, IBATIS evita que el usuario se encargue de buscar la fuente de datos, administrar el pool de conexiones, etc. ya que lo hace él.

### 3.2.2. *HIBERNATE*

HIBERNATE es una herramienta ORM para Java. Permite al usuario hacer persistentes objetos, siguiendo las propiedades del lenguaje Java tales como asociaciones, herencia, polimorfismo, composición y todo lo que son Java Collections.

No solo provee relaciones entre clases Java y tablas de base de datos, sino también provee recuperación de datos a través de un mecanismo propietario de consulta llamado HQL (Hibernate Query

Language). Todo esto permite reducir los tiempos de desarrollo de manera drástica evitando el manejo manual de SQL y JDBC.

El objetivo principal de HIBERNATE es la eliminación del programador de toda tarea común referente a la persistencia de los datos.

Este framework hace uso de clases POJO (Plain Old Java Object), que junto a ficheros xml permiten relacionar esos objetos con la base de datos relacional. Los objetos que se deseen hacer persistentes, se deben definir en un documento de mapping. Este documento de mapping es compilado en el momento de inicialización de la aplicación y proveen al framework de la información necesaria para cada clase.

Otra característica importante de HIBERNATE, es que puede generar un esquema de base de datos a raíz del modelo de clases y los documentos de mapeo y viceversa.

### 3.2.3. *EJB (Enterprise Java Beans)*

Los EJB son componentes que representan los datos persistentes del modelo de negocio de la aplicación. Se trata de una representación Java, en memoria y en forma de objeto de la información almacenada en algún medio persistente.

Dentro de los EJB, hay dos maneras diferentes de gestionar la persistencia de objetos:

- El desarrollador especifica el comportamiento de la persistencia, estaremos hablando de EJB de entidad con persistencia gestionada por el componente (BMP, Bean Managed Persistence).
- El contenedor especifica el comportamiento de la persistencia, estaremos hablando de EJB de entidad con persistencia gestionada por el contenedor (CMP, Container Managed Persistence).

En los BMP (EJB de entidad con persistencia gestionada por el componente), el desarrollador del componente es la persona que debe implementar la persistencia de dicho componente. Esto quiere decir que el desarrollador debe especificar cada uno de los diferentes comportamientos que tendrá el componente en la comunicación con la base de datos relacional. Este tipo de EJB es mucho más costoso para el desarrollador y puede llevar a un número mayor de errores de persistencia.

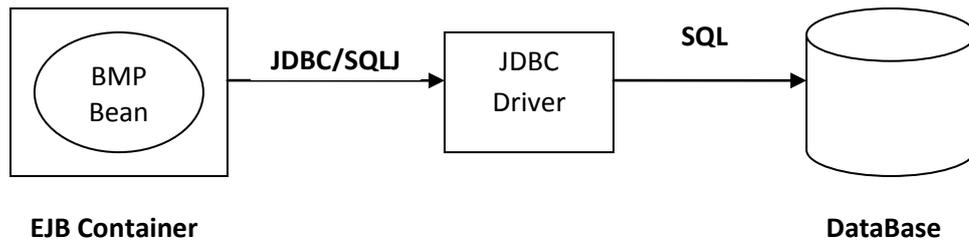


Ilustración 4.- Esquema BMP

En los CMP (EJB de entidad con persistencia gestionada por el contenedor), el desarrollador no tiene que programar la persistencia del componente. El desarrollador tiene que definir los campos y las relaciones persistentes del componente que se quiere desarrollar, y el contenedor se encarga de generar el código de gestión de la persistencia según esta definición cuidadosa en tiempo de despliegue del componente.

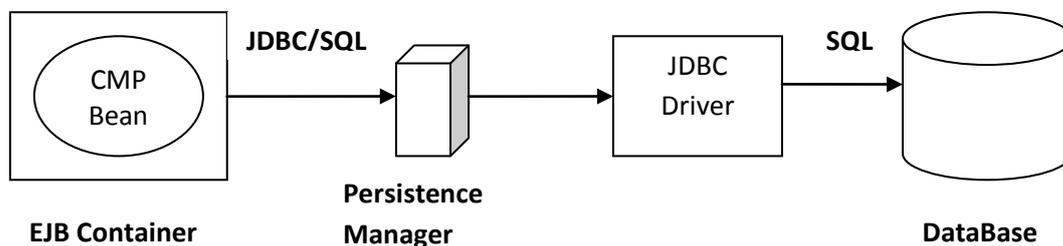


Ilustración 5.- Esquema CMP

El lenguaje utilizado por este framework para definir la persistencia es muy similar al SQL y se denomina EJB QL. Este lenguaje se escribe con la misma estructura de un xml en el descriptor de despliegue de la aplicación y permite a los usuarios describir el comportamiento de métodos de consulta de una manera transparente, haciendo estas consultas transportables a bases de datos de diferentes fabricantes.

### 3.2.4. JDO (Java Data Object)

JDO, como su nombre bien indica es una especificación Java destinada a la administración de las bases de datos orientadas a objetos. El creador de esta especificación es Sun, el cual suministra una serie de librerías que sirven como complemento al trabajo del desarrollador.

No exige la implementación de interfaces de desarrollo. La configuración de este framework se realiza mediante un XML. JDO permite

trabajar con objetos normales de Java a la vez que los hace persistentes de forma que requiere de una menor estructura.

### *3.2.5. Comparación de los diferentes frameworks de persistencia*

Para realizar la comparación de los frameworks de persistencia detallados anteriormente vamos a seguir unos criterios generales de forma que de cara a un desarrollador se pueda obtener el más adecuado a cada proyecto. Estos criterios son los siguientes:

- **Portabilidad:** En el caso de Hibernate, al ser JBOSS el único proveedor de este mecanismo, obliga que todos los proyectos que se vayan a ejecutar en servidores de aplicaciones diferentes a este deban tener incluidas las librerías de Hibernate. Mientras que en el resto de marcos de trabajo, no habría que modificar la aplicación para que se adapte a los diferentes servidores.
- **Curva de aprendizaje:** Todos los frameworks de persistencia tratados tienen una curva de aprendizaje relativamente corta, aunque por ejemplo en Hibernate son pocos los profesionales que utilizan su potencial al máximo.
- **Funcionalidad:** En este caso la principal diferencia es que tanto Hibernate, Ibatis como JDO están únicamente orientados a la capa de persistencia, mientras que EJB es un marco de trabajo para el desarrollo de aplicaciones empresariales en JAVA.
- **Escalabilidad:** Todos los marcos de trabajo son escalables.
- **Generación de código SQL:** Todos a excepción de Ibatis generan el código SQL de forma propia aunque soportan la ejecución de código SQL por parte del programador de la aplicación.
- **Opciones de trabajo:** Los datos aportados son mediante la consulta a la oferta actual en el portal web de empleo "Infojobs".

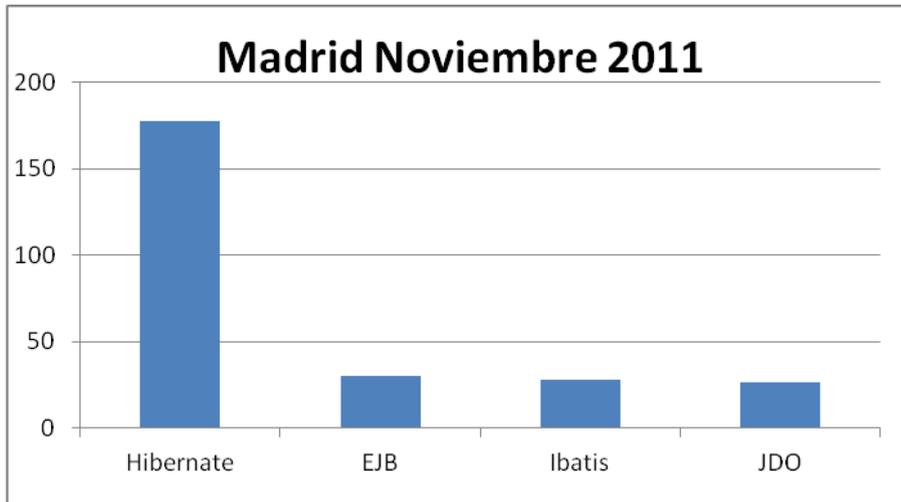


Ilustración 6.- Gráfica opciones de trabajo J2EE

- Cantidad de soporte: Para medir la cantidad de soporte disponible, haremos una consulta a la empresa "la casa del libro", por su disponibilidad en las grandes ciudades.

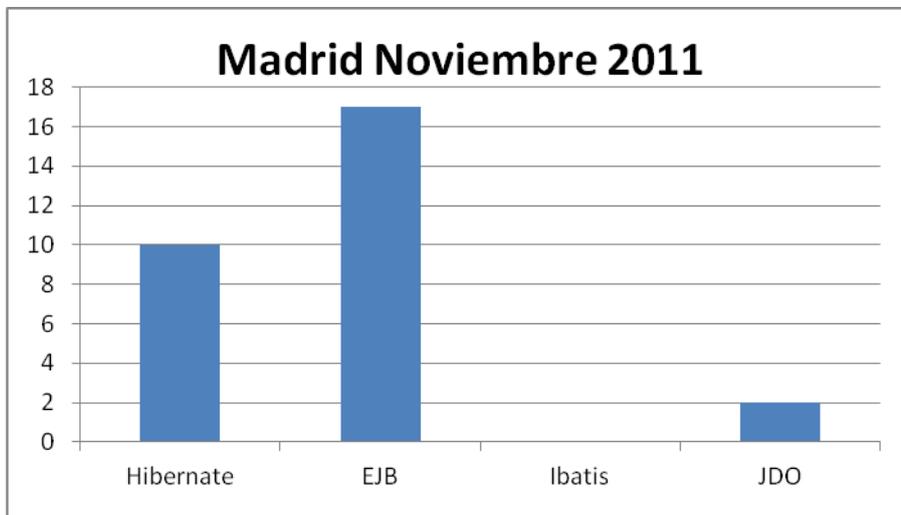


Ilustración 7.- Gráfica cantidad de soporte J2EE

## 4. Diseño de un framework de persistencia

### 4.1. *Introducción*

Básicamente el objetivo de este framework de persistencia es ofrecer al usuario un interfaz de interacción con una base de datos relacional, sin que este tenga que hacer modificaciones, únicamente indicando el tipo de base de datos a la que se accederá.

### 4.2. *Características*

Las características que debe tener el framework de persistencia que vamos a desarrollar a lo largo de este proyecto serán las siguientes:

- Únicamente tendrá dependencia con las clases de la JDK de Java, ya que es este lenguaje en el que se implementará.
- Debe ser independiente a cualquier tipo de base de datos relacional.
- Para establecer las consultas a BBDD, se usará JDBC 1.x o superior.
- Deberá poder soportar la implementación de varias clases por cada tabla de la base de datos.
- Realizaremos el mapeo mediante fichero de propiedades.

### 4.3. *Mapeo*

Por cada uno de los objetos que tengamos en la aplicación, deberemos tener un fichero de propiedades, el cual sirva para mapear la entidad de base de datos con el objeto java. Estas variables tendrán la siguiente sentencia:

*Nombre\_tabla.campo = campo*

## 4.4. Análisis y Diseño del framework

### 4.4.1. Conexión y relación de la Base de Datos

Esta es una parte muy importante del proyecto, ya que es sobre la base de datos donde recaerá toda la funcionalidad final. Es importante definir que en este framework se van a relacionar las tablas con los objetos con los que trabajemos.

Independientemente del nombre del objeto que creemos, este deberá guardar correspondencia con un fichero de properties, que será el encargado de definir el mapeo de las tablas de Base de Datos con los atributos de las entidades que vayamos a utilizar. Además, este properties se utilizará para poder obtener la información necesaria para la construcción de las sentencias SQL y las cadenas de conexión. Cada tabla tendrá su properties en concreto y cada una de las entradas de dicho properties, tendrá la siguiente estructura:

*nombre\_tabla.nombre\_campo = etiqueta\_campo*

Puesto que estamos hablando en este apartado de la Base de Datos con la que se interrelacionará el framework de persistencia, es necesario que hagamos mención de la conexión que utilizamos y como configurarla para diferentes sistemas. El sistema elegido es, al igual que lo comentado anteriormente, gestionar la configuración de la conexión mediante fichero de properties. En concreto el fichero de properties debe estar compuesto por las siguientes variables:

- nombreDriver={nombre del driver que nos permitirá establecer conexión con la Base de Datos}
- dbURL={cadena de conexión con la Base de Datos}
- nombreUsuario={nombre de usuario con el que acceder a la Base de Datos}
- password={contraseña del usuario introducido anteriormente}

Por defecto se ha definido dentro del framework de persistencia que el fichero de configuración se encuentre alojado en la siguiente ruta física *C:/conf/conexion.properties*.

La clase encargada de toda esta gestión es *DataSourceBasicoImpl*, que se encuentra ubicada en el paquete *dao*.

Este es el constructor básico en el que se define la conexión que por defecto se va a utilizar en el acceso del framework a la base de datos relacional:

```

public DataSourceBasicoImpl() throws FWPersistenciaExceptionImpl {
    try {
        propiedadesConexion.load(new
        FileInputStream("C:/conf/conexion.properties"));
    } catch (FileNotFoundException e) {
        throw new FWPersistenciaExceptionImpl("No existe el fichero
necesario para la carga de propiedades de la conexión");
    } catch (IOException e) {
        throw new FWPersistenciaExceptionImpl("El fichero con las
propiedades de conexión no es válido");
    }

    nombreDriver = propiedadesConexion.getProperty("nombreDriver");
    dbURL = propiedadesConexion.getProperty("dbURL");
    nombreUsuario = propiedadesConexion.getProperty("nombreUsuario");
    password = propiedadesConexion.getProperty("password");
}
}

```

Y esta es la representación de la clase completa, con sus atributos y métodos:

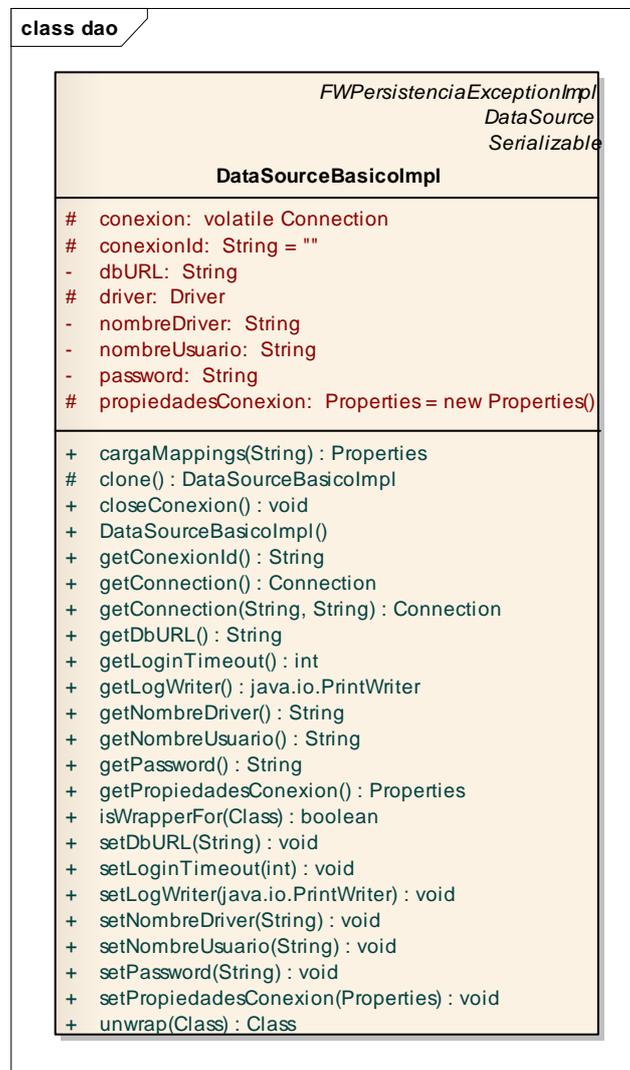


Ilustración 8.- Clase DataSourceBasicoImpl

#### 4.4.2. Sistema de excepciones

En el desarrollo del framework, se ha considerado oportuno la gestión de excepciones creando un nuevo tipo de excepción propio. Este tipo de excepciones nos servirán para devolver al usuario de forma controlada los posibles errores (con sus causas) que se puedan dar durante la utilización del framework.

El sistema implementado, es muy básico teniendo un único constructor de la clase con información, de esta forma se podrá indicar al usuario mediante una descripción, el error que se ha producido, en que función del código y la causa de este.

La clase destinada a este cometido es `FWPersistenciaExceptionImpl`, y se encuentra ubicada en el paquete `exception`.

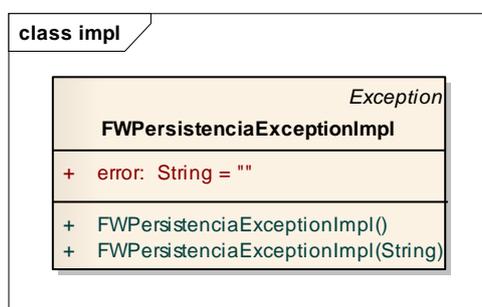


Ilustración 9.- Clase `FWPersistenciaExceptionImpl`

#### 4.4.3. Diagrama de paquetes

Para simplicidad de implementación, es recomendable el agrupar las clases que forman parte del framework en paquetes según la funcionalidad. Los paquetes de los que se compondrá nuestro framework son los siguientes:

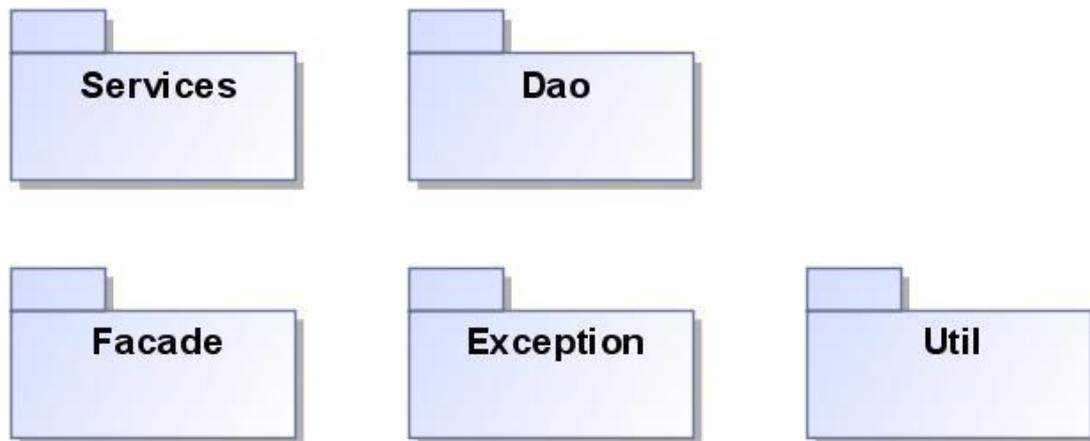


Ilustración 10.- Diagrama de paquetes

A continuación haremos una breve descripción de cada una de las funcionalidades que comprenderán:

- Services: clases que gestionan la lectura de la información de los diferentes objetos, así como las clases que generan automáticamente las consultas a través de los objetos.
- Dao: clases que establecen la conexión con la base de datos y ejecutan las consultas haciendo uso de las clases del paquete anterior.
- Actions: Clases que son utilizadas por las clases del paquete facade para ejecutar sus acciones.
- Facade: clases a través de las cuales se accede al framework.
- Exception: clases que nos gestionan las posibles excepciones que se puedan producir en el framework.
- Util: clases de utilidades.

#### 4.4.4. Diagrama de secuencias

En este apartado analizaremos los diferentes pasos por los que pasa el framework en cada una de las operaciones que facilita al usuario y que por lo tanto para este serán transparentes.

Un diagrama básico que nos facilita la secuencia de paquetes es el siguiente:

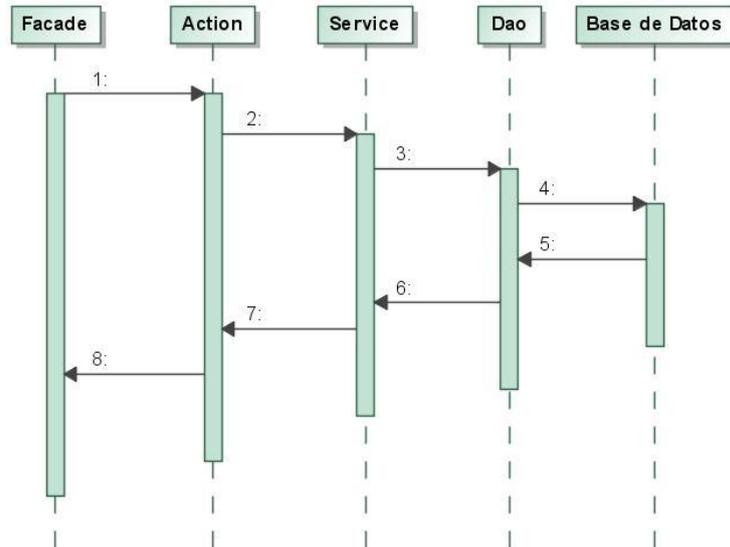


Ilustración 11.- Diagrama de secuencias

Con este diagrama podemos comprobar cómo sería la interrelación entre las clases de los diferentes paquetes que forman el framework. Este diagrama es global, puesto que cada funcionalidad en concreto tendrá su diagrama de secuencias propio.

A continuación veremos las diferentes secuencias del framework. Como podemos observar la tramitación de la información es totalmente transparente para el usuario.

#### 4.4.4.1. Secuencia de creación

En este apartado se representa de forma gráfica y secuencial los diferentes pasos por los que pasa la funcionalidad de creación. Cabe destacar que se han marcado las principales transacciones entre los diferentes paquetes que forman el framework de persistencia desarrollado.

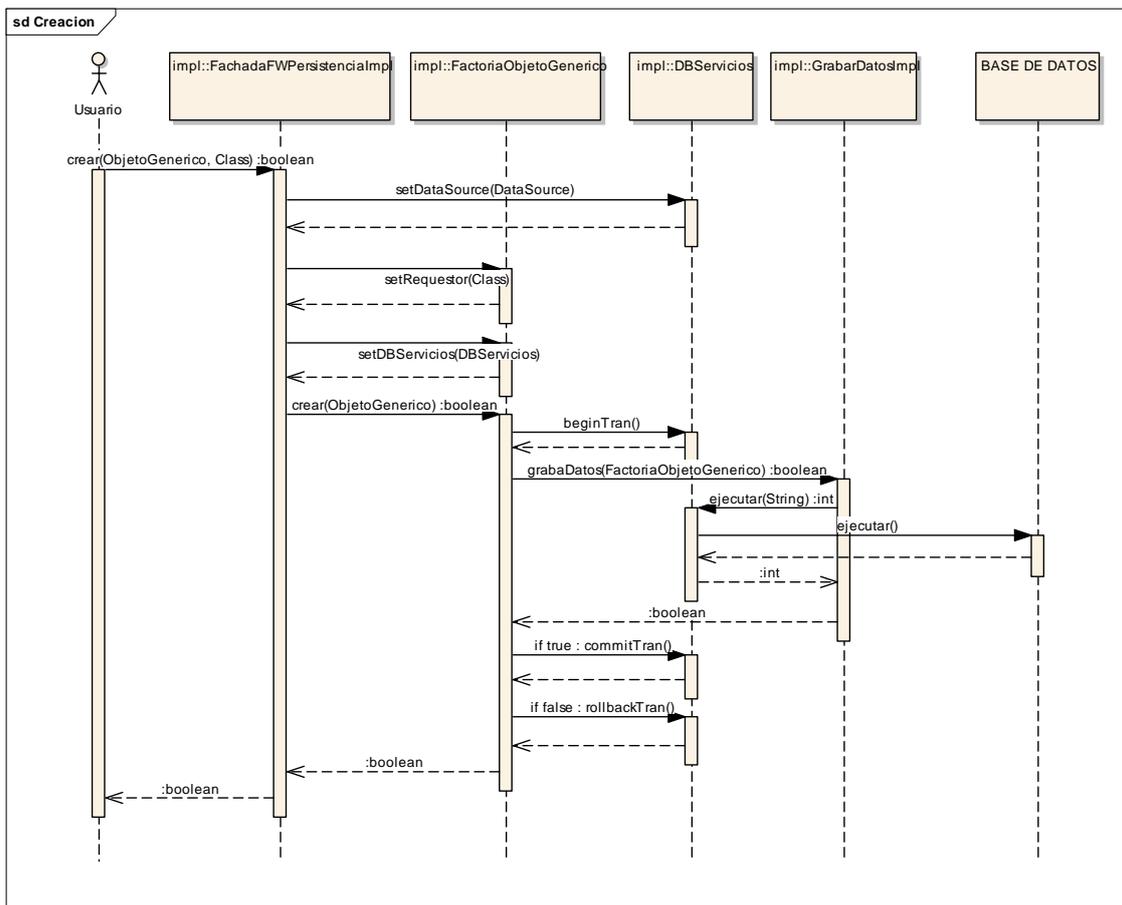


Ilustración 12.- Secuencia de creación

#### 4.4.4.2. Secuencia de eliminación

En esta secuencia al igual que la anterior analizaremos los principales movimientos de información entre las clases del framework.

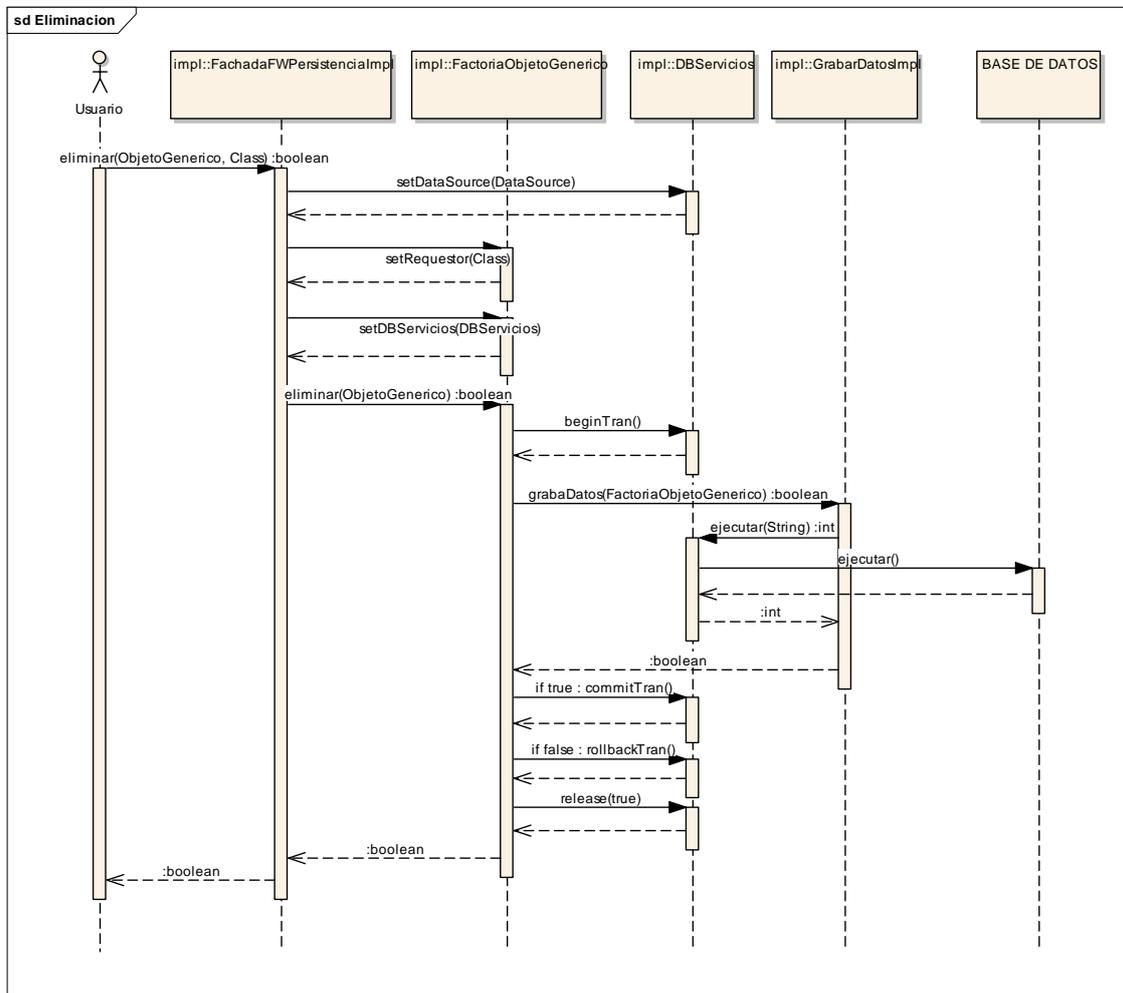


Ilustración 13.- Secuencia de eliminación

#### 4.4.4.3. Secuencia de consulta

Esta secuencia es mucho más rápida e intuitiva, puesto que únicamente es de consulta y no es necesario la adaptación de la información para que sea ejecutable contra la base de datos.

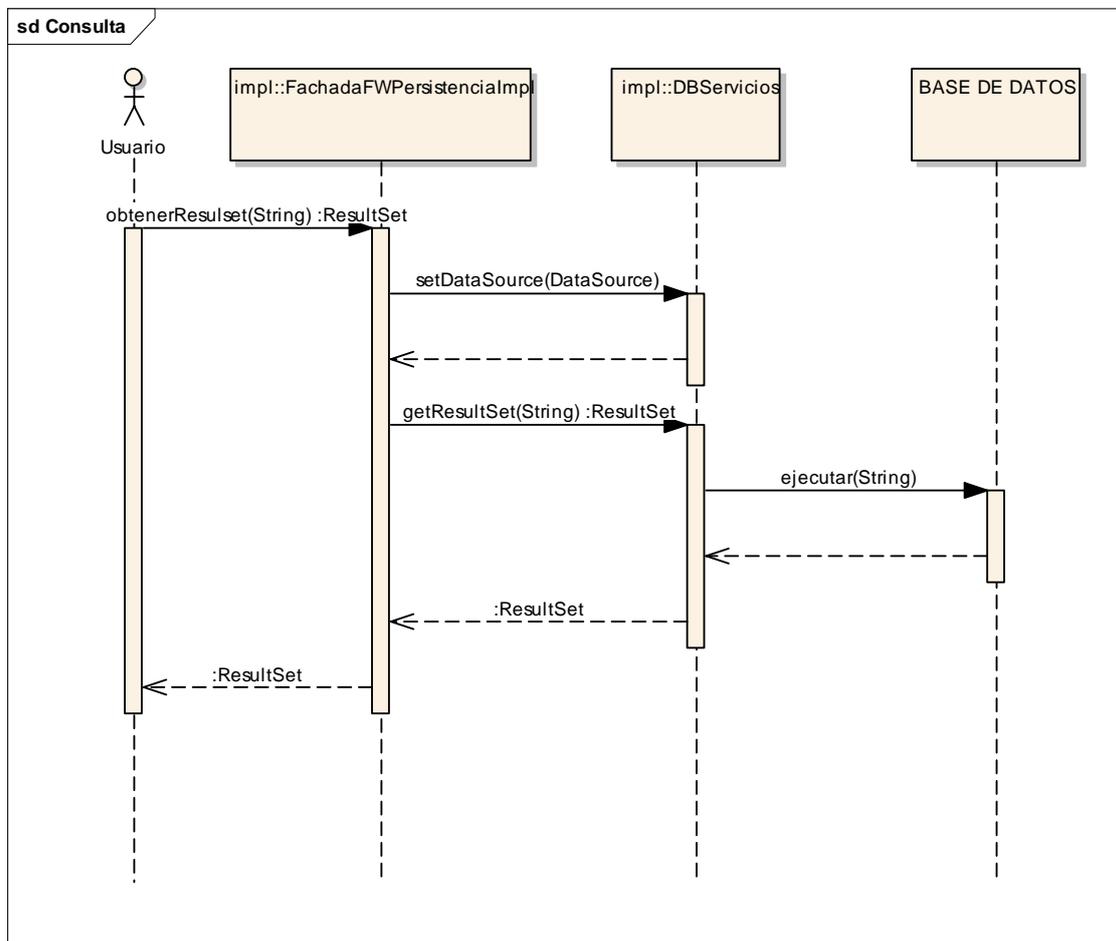


Ilustración 14.- Secuencia de consulta

#### 4.4.4.4. Secuencia de modificación

Por último tenemos la de modificación, es importante resaltar la gestión interna de la información que se hace y que no sale en estos diagramas por claridad, ya que en la clase GrabarDatosImpl se hace gran parte de la lógica de negocio del framework de persistencia.

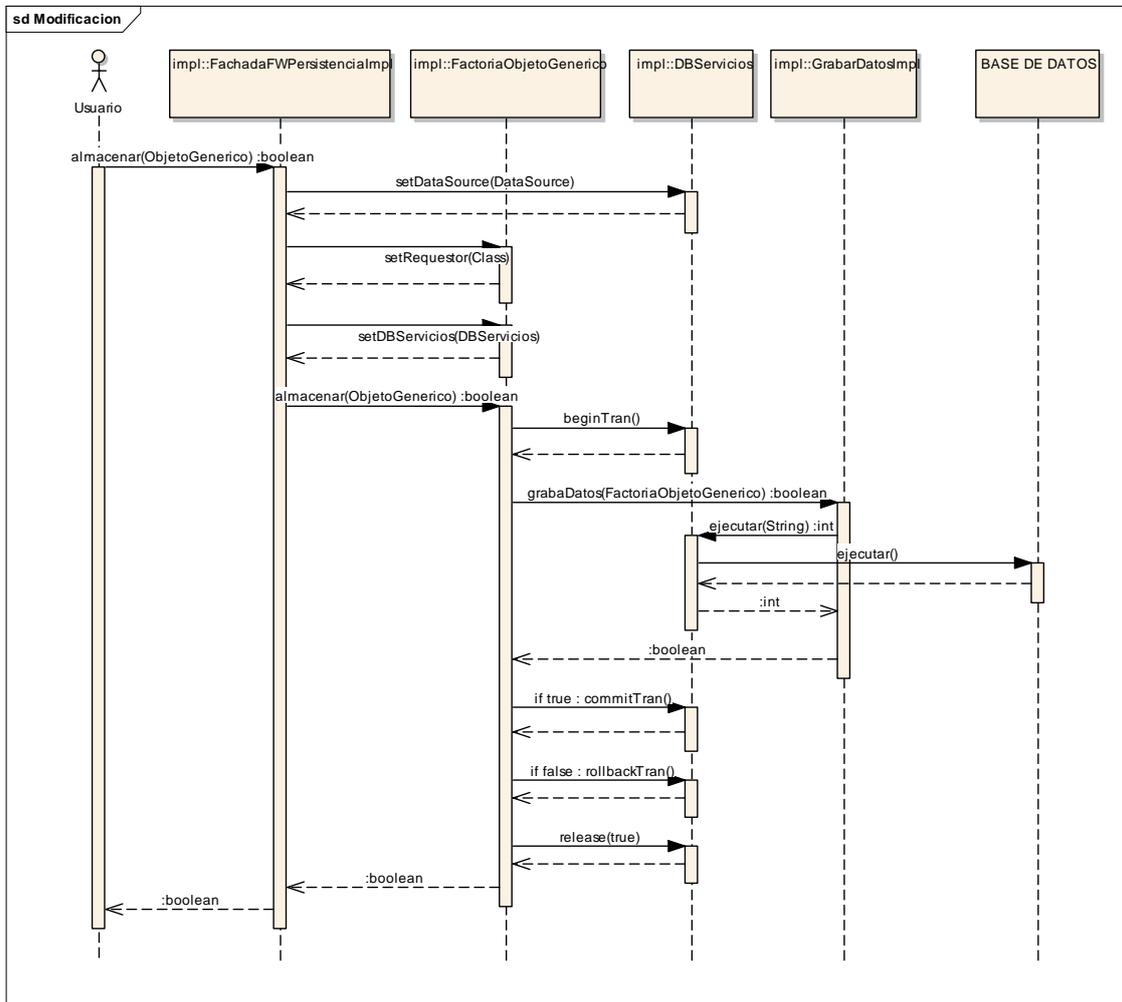


Ilustración 15.- Secuencia de modificación



## 5. Aplicación de ejemplo

### 5.1. Introducción

Para comprobar el correcto funcionamiento del framework y de las operaciones que tiene al servicio del usuario, hemos implementado una pequeña aplicación web que utilizando las funciones del framework, realizará un pequeño mantenimiento sobre la base de datos.

El framework, está preparado para poder utilizarse dentro de cualquier aplicación en la que se utilice java o que sea compatible con esta tecnología. Se ha optado por una aplicación web, puesto que nos permite con mayor comodidad poder comprobar los resultados e interactuar con ellos.

La aplicación se ha creado apoyándose en la tecnología JSP. He elegido esta tecnología, ya que me permite desarrollar tanto la parte visual como la parte de negocio de la aplicación sobre un mismo fichero. Es por ello que de esta forma con un único fichero, visualizaremos la información y a la vez estaremos comunicándonos con el framework.

Para la Base de Datos relacional se ha optado por MySQL y por lo tanto debemos instalarnos el driver necesario para esta base de datos.

El modelo sobre el que se van a realizar las operaciones de prueba es el siguiente:

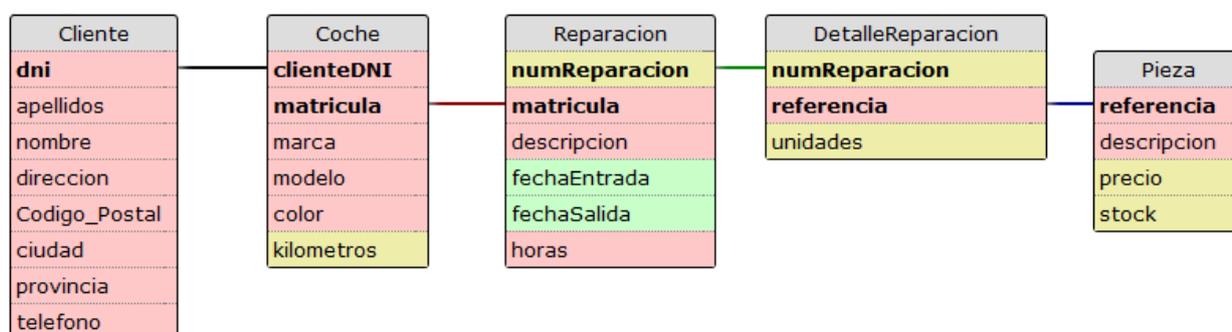


Ilustración 17.- Modelo de datos aplicación de ejemplo

Se trata de un modelo de datos muy sencillo, puesto que el objetivo de esta aplicación no es la de pruebas de estrés o de complejidad, sino la de comprobar que los servicios que habilita el framework para trabajar con la Base de Datos, funcionan correctamente.

Como hemos hablado de la sencillez del modelo de datos, vamos a añadir que las pruebas se han realizado únicamente con la tabla cliente, puesto que el resto de tablas funcionan de la misma manera y por lo tanto la clase que utilizaremos es ClienteUOC.

## *5.2. Uso de la aplicación*

Para poder ejecutar la aplicación, tendremos que acceder a la siguiente ruta: <http://localhost:8080/prototipo>, a partir de la cual, se pueden ir ejecutando las operaciones sobre el modelo de datos.

### *5.2.1. Listado de clientes*

Esta operación recupera el contenido de la tabla Cliente y muestra sus datos en un listado como el siguiente:

LISTADO GLOBAL - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

LISTADO GLOBAL

localhost:8080/prototipo/listado.jsp

Más visitados Comenzar a usar Fire... Últimas noticias Personalizar vínculos Acceso clientes. Acce... Trac Incidencias PRE Mantis

**LISTADO DE CLIENTES**

DNI	Apellidos	Nombre	Dirección	Código Postal	Ciudad	Provincia	Teléfono
52355354V	Barrero Muñoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121
78714561H	Villamil Lozano	Victoria	c/ PexasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754

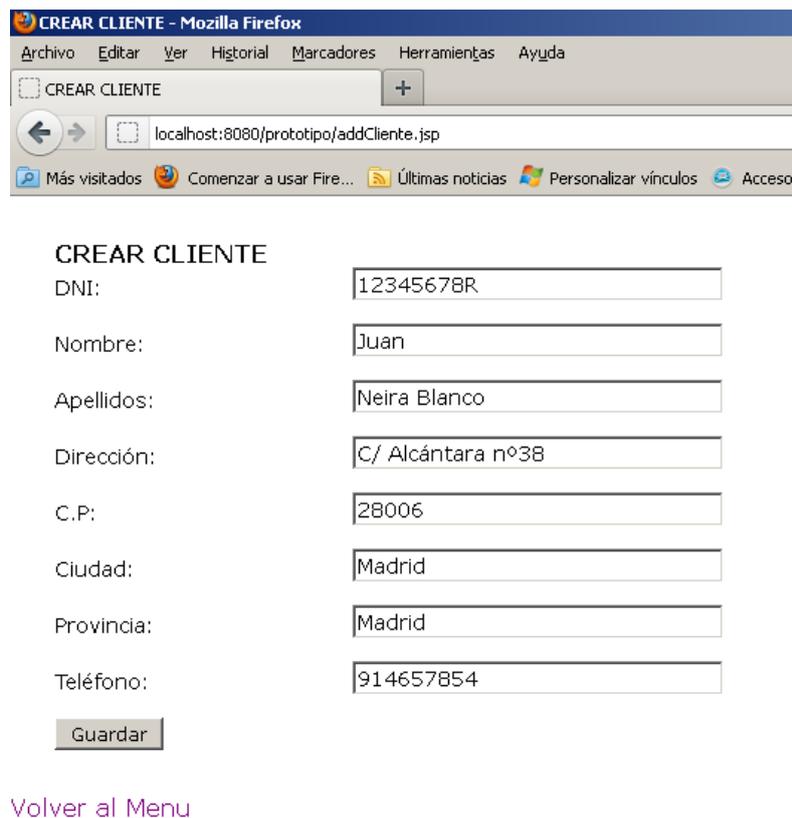
[Volver al Menu](#)

Ilustración 18.- Pantalla de listado de la aplicación de ejemplo

Este listado se recupera a partir de una sentencia SQL que se introduce desde la página JSP.

### 5.2.2. *Añadir un nuevo cliente*

Operación que nos permite dar de alta un nuevo cliente en su correspondiente tabla.



The screenshot shows a Mozilla Firefox browser window with the title 'CREAR CLIENTE - Mozilla Firefox'. The address bar shows 'localhost:8080/prototipo/addCliente.jsp'. The form contains the following fields and values:

CREAR CLIENTE	
DNI:	12345678R
Nombre:	Juan
Apellidos:	Neira Blanco
Dirección:	C/ Alcántara nº38
C.P.:	28006
Ciudad:	Madrid
Provincia:	Madrid
Teléfono:	914657854
<input type="button" value="Guardar"/>	

[Volver al Menu](#)

**Ilustración 19.- Pantalla de creación de cliente**

A continuación mostraremos el listado de los clientes una vez realizada la creación de uno nuevo.

Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

http://localhost:8080...d&telefono=914657854 +

localhost:8080/prototipo/resultadoAddCliente.jsp?dni=12345678R&nombre=Juan&apellidos=Neira+Blanco&direccion=C%2F+Alcntara+n38&codigo\_postal=

Ms visitados Comenzar a usar Fire... ltimas noticias Personalizar vnculos Acceso clientes. Acce... Trac Incidencias PRE Mantis

El elemento se ha creado correctamente.

**LISTADO DE CLIENTES**

DNI	Apellidos	Nombre	Direccin	Codigo Postal	Ciudad	Provincia	Telefono
12345678R	Neira Blanco	Juan	C/ Alcntara n38	28006	Madrid	Madrid	914657854
52355354V	Barrero Muoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121
78714561H	Villamil Lozano	Victoria	c/ PexasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754

[Volver al Menu](#)

Ilustracin 20.- Listado de clientes (Creacin)

### 5.2.3. Modificar un cliente ya existente

Para el desarrollo de esta función, primero debemos partir de un elemento correcto desde la BD y una vez mostrado modificarlo para que el framework de persistencia lo detecte y realice una actualización de ese registro en Base de Datos.



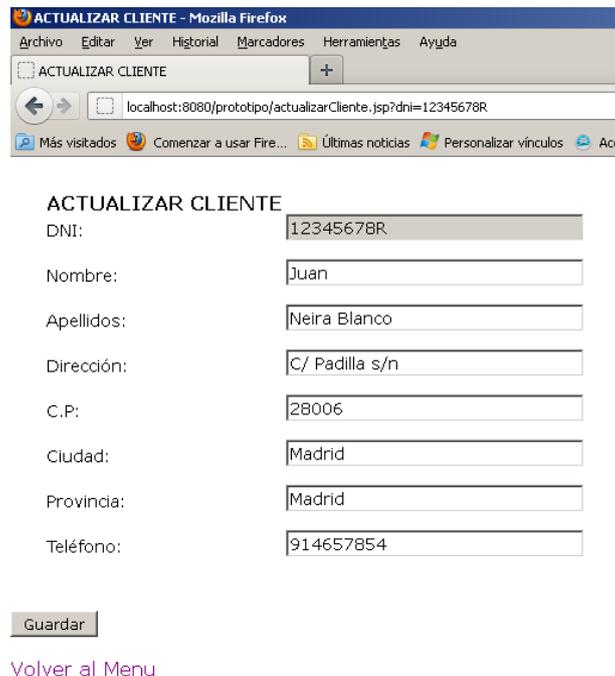
ACTUALIZACIÓN DE CLIENTES

DNI	Apellidos	Nombre	Dirección	Codigo Postal	Ciudad	Provincia	Telefono	Selección
12345678R	Neira Blanco	Juan	C/ Alcántara nº38	28006	Madrid	Madrid	914657854	<input checked="" type="radio"/>
52355354V	Barrero Muxoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121	<input type="radio"/>
78714561H	Villamil Lozano	Victoria	C/ PexasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651	<input type="radio"/>
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754	<input type="radio"/>

Actualizar  
[Volver al Menu](#)

Ilustración 21.- Listado previo a la modificación de un cliente

Una vez que hemos realizado la selección del elemento que vamos a modificar, al pulsar en el botón de actualizar la aplicación nos mostrará los datos del cliente y procederemos a la modificación de este.



ACTUALIZAR CLIENTE

DNI:

Nombre:

Apellidos:

Dirección:

C.P:

Ciudad:

Provincia:

Teléfono:

Guardar  
[Volver al Menu](#)

Ilustración 22.- Formulario de modificación

Mozilla Firefox

http://localhost:8080...dstelefono=914657854

localhost:8080/prototipo/resultadoUpdateCliente.jsp?dni=12345678R&nombre=Juan&apellidos=Neira+Blanco&direccion=C%2F+Padilla+s%2Fn&codigo\_post

Más visitados Comenzar a usar Fire... Últimas noticias Personalizar vínculos Acceso clientes, Acce... Trac Incidencias PRE Mantis

El elemento se ha modificado correctamente.

**LISTADO DE CLIENTES**

DNI	Apellidos	Nombre	Dirección	Codigo Postal	Ciudad	Provincia	Telefono
12345678R	Neira Blanco	Juan	C/ Padilla s/n	28006	Madrid	Madrid	914657854
52355354V	Barrero Muñoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121
78714561H	Villamil Lozano	Victoria	c/ PeñasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754

Volver al Menu

**Ilustración 23.- Listado tras la modificación de un cliente**

#### 5.2.4. Eliminar un cliente

Se realiza la eliminación de un registro de la Base de Datos.

ELIMINAR CLIENTE - Mozilla Firefox

ELIMINAR CLIENTE

localhost:8080/prototipo/eliminarCliente.jsp

Más visitados Comenzar a usar Fire... Últimas noticias Personalizar vínculos Acceso clientes, Acce... Trac Incidencias PRE Mantis

**LISTADO DE CLIENTES**

DNI	Apellidos	Nombre	Dirección	Codigo Postal	Ciudad	Provincia	Telefono	Selección
12345678R	Neira Blanco	Juan	C/ Padilla s/n	28006	Madrid	Madrid	914657854	<input checked="" type="radio"/>
52355354V	Barrero Muñoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121	<input type="radio"/>
78714561H	Villamil Lozano	Victoria	c/ PeñasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651	<input type="radio"/>
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754	<input type="radio"/>

Eliminar

Volver al Menu

**Ilustración 24.- Listado previo a la eliminación de un cliente**

Al igual que hemos realizado en la modificación, es necesario que seleccionemos un cliente a eliminar. De esta forma la aplicación se comunicará con el framework de persistencia y se procederá a la eliminación del registro en la base de datos relacional.

El elemento se ha borrado correctamente.

**LISTADO DE CLIENTES**

DNI	Apellidos	Nombre	Dirección	Código Postal	Ciudad	Provincia	Teléfono
52355354V	Barrero Muñoz	Antonio	C/ Curro Romero 4	06400	Don Benito	Badajoz	924815121
78714561H	Villamil Lozano	Victoria	c/ PexasArriba s/n	13003	Ciudad Real	Ciudad Real	926226651
78715889E	Hidalgo Latorre	Juan	C/ Ayala 23	28006	Madrid	Madrid	914548754

[Volver al Menu](#)

**Ilustración 25.- Listado de clientes tras la eliminación**

### 5.3. Instalación de la aplicación

Para proceder al uso de la aplicación de ejemplo desarrollada, debemos instalar previamente el software necesario para poder realizar las pruebas de funcionamiento.

#### 5.3.1. Java Runtime Environment (JRE)

Para el desarrollo de la aplicación se ha utilizado la versión 5.0, para poder ejecutar el servidor de aplicaciones y poder utilizar correctamente el framework desarrollado, deberemos tener instalado en la máquina el JRE versión 5.0 o superior.

Esta versión se puede descargar de <http://www.oracle.com/technetwork/java/javase/downloads/index.html> e instalarlo según indique el wizard de instalación. Una vez descargado e instalado, será necesario definirlo como una variable de entorno para que esté visible para cualquier aplicación del equipo que lo necesite.

#### 5.3.2. Tomcat

La aplicación se ha probado en el tomcat como servidor de aplicaciones. Este puede ser descargado desde <http://tomcat.apache.org/download-60.cgi>.

Una vez descargado se descomprime el fichero en el disco duro. Tras esta operación se habrán creado una serie de directorios entre los que cabe destacar los siguientes:

- Bin: Directorio con los ficheros ejecutables para arrancar y parar el servidor.
- Doc: Documentación de la distribución en formato html.
- Conf: Ficheros de configuración.
- Webapps: Contien diversas aplicaciones web de ejemplo. Es en este directorio, donde debemos depositar la aplicación de ejemplo.

Dentro de la carpeta bin, nos encontraremos el fichero de arranque (startup.bat) y el fichero de parada (shutdown.bat).

### 5.3.3. MySQL

MySQL es un sistema de administración de bases de datos muy potente. La principal virtud es que es totalmente gratuito, por lo que es una fuerte alternativa ante sistemas como SQL u Oracle. Además, cumple los requerimientos mínimos para mostrar el funcionamiento del framework. Cabe decir que el framework se puede utilizar con cualquier motor de base de datos para el que exista driver JDBC.

Para proceder a la instalación es necesario descargar en el sitio oficial de MySQL la versión correspondiente al sistema operativo que se vaya a utilizar. MySQL se puede descargar de <http://dev.mysql.com/downloads/>.

La manera más sencilla es mediante un instalable, de forma que una vez que se lanza simplemente hay que seguir las indicaciones que nos marque la aplicación.

### 5.3.4. Instalación de la aplicación

Una vez que tenemos todos los componentes de la arquitectura instalados, sólo resta la instalación y configuración de la aplicación de ejemplo. Para ello seguiremos los siguientes pasos:

- Lo primero es situar en la carpeta webapps de Tomcat el fichero prototipo.war con la aplicación de ejemplo.
- Al iniciar Tomcat, ya tendremos la aplicación desplegada, pero nos faltará definir la información que debe haber en la Base de Datos. Para ello habrá que ejecutar un script SQL. Para la importación de estos datos, se puede hacer directamente por consola de MySQL o desde algún programa que permita la conexión y la ejecución de ese tipo de sentencias.

- Por último, se ha de verificar los parámetros de conexión de la aplicación web con la base de datos. Tendremos que editar los valores del fichero "conexion.properties". Estos valores son los expuestos en el punto 4.3.1 del presente documento.

Una vez que esté todo instalado y configurado, al acceder a la aplicación, obtendremos la siguiente pantalla:

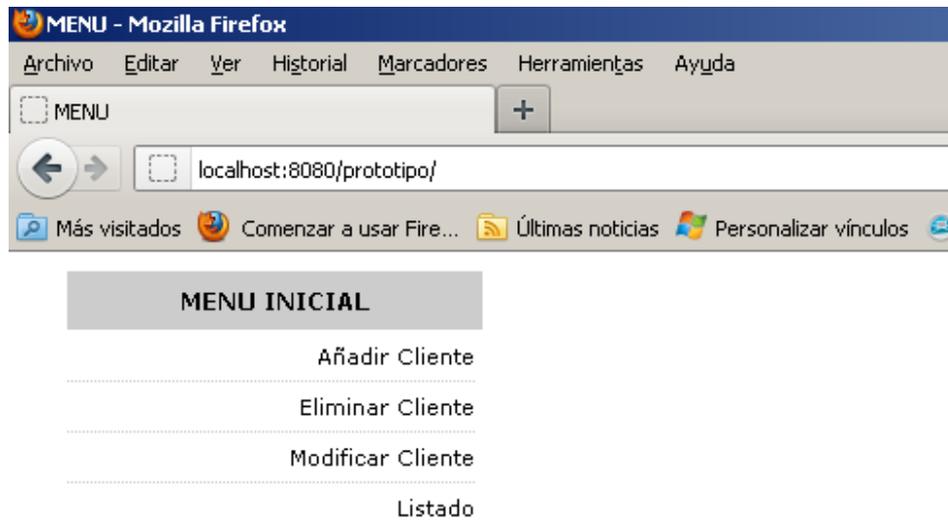


Ilustración 26.- Pantalla inicial de la aplicación

## 6. Conclusiones

### 6.1. *Objetivos cumplidos*

En la introducción se plantearon una serie de objetivos del proyecto a completar durante la realización del mismo que se han cumplido tanto en profundidad como en plazos de tiempo.

La realización de este proyecto me ha permitido poder estudiar a fondo el problema de la persistencia dentro de aplicaciones java y la correspondencia entre el modelo de datos relacional y el modelo de objetos java.

Así mismo, me ha dado la oportunidad de conocer a un poco más a fondo las APIs que suministra la tecnología JAVA, en concreto las de la versión 1.5 que ha sido la versión elegida para utilizar durante el desarrollo del framework.

### 6.2. *Ampliaciones del framework*

El framework desarrollado únicamente cubre las operaciones básicas a realizar sobre una base de datos (creación, modificación, eliminación y consulta), además de que únicamente se ha probado en modelos de datos muy básicos y que presentan poca complejidad.

Las pruebas que se han realizado sobre el framework se han realizado sobre un modelo de pequeñas dimensiones, tipos de datos y con características que han garantizado el correcto funcionamiento. Si se utilizasen modelos de datos de una complejidad considerable puede ser que el framework no se comporte de la manera deseada.

A continuación, se enumerarán una serie de ampliaciones que se pueden hacer al framework para que este funciones de una manera más estable y que un número mayor de funcionalidades:

- **Creación de sentencias:** Una de las posibles mejoras es la de la creación de sentencias SQL más complejas, de forma que nos permita sacarle mayor partido a la base de datos relacional con la que estemos trabajando.

- **Recursividad:** Es una de las mejoras que se le podrían aplicar al framework. Esta mejora requiere de un estudio mucho más profundo de cara a controlar todas las diferentes opciones que pueden darse en una base de datos relacional.

## 7. Glosario

- **API:** Es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizados por otro software como una capa de abstracción.
- **Atributo de una entidad:** Propiedad que forma parte de una entidad.
- **Base de Datos:** Es la representación íntegra de los conjuntos de entidades o instancias de información y sus interrelaciones.
- **Capa:** Es cada una de las partes en las que se divide una aplicación que cumpla el patrón MVC. Este framework sería la capa que accede a la Base de datos para todas las aplicaciones que utilicen este framework.
- **Clase:** Tipo de datos definido por el usuario que especifica un conjunto de objetos que comparten las mismas propiedades.
- **Consulta:** Sentencia realizada a una base de datos, en la que se pide información sobre unos criterios concretos.
- **Commit:** Término que define la acción de confirmar la transacción actual. Todos los cambios realizados en la transacción se trasladan de forma permanente en la base de datos.
- **Driver JDBC:** Implementación particular del JDBC para una base de datos en concreto.
- **Entidad:** Es un objeto que se distingue de otros objetos por medio de un conjunto específico de atributos.
- **Excepción:** Término propio de los lenguajes de programación, y hace referencia a los errores que se pueden producir durante la ejecución de una aplicación.
- **Herencia de objetos:** Una de las tres propiedades principales de los objetos en Java. Permite generar clases y objetos a partir de otros ya existentes, manteniendo algunas o todas sus propiedades.
- **Implementación:** Hace referencia a programar una aplicación a partir de un modelo de aplicación teórico o esquemático.
- **Interrelación:** Asociación entre entidades.
- **Java:** Lenguaje de programación orientado a objetos, creado por Sun para generar aplicaciones exportables a Internet.
- **J2EE:** Paquete del lenguaje Java que recoge sobre todo librerías orientadas a servidores de aplicaciones.
- **JDBC:** Es un API estándar de Java para interactuar con las abstracciones y conceptos de las BD relacionales y, en concreto, para trabajar con SQL.
- **MVC:** es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

- **Modelo de datos:** Término que hace referencia al sistema formal y abstracto que nos permite describir los datos a partir de una serie de reglas y convenios predeterminados.
- **Objeto:** Es la representación de un objeto real en Java. Es producto de una Clase y cumple las propiedades del objeto real y que son necesarias para nuestra aplicación. Se compone de atributos, métodos y relaciones con otros objetos.
- **Persistencia:** Propiedad que hace referencia al hecho de que los datos sobre los que trata un programa no se pierdan al acabar su ejecución.
- **Rollback:** La traducción literal es "marcha atrás". Nos permite recuperar el estado anterior a una operación sobre la información en una base de datos.
- **SQL:** Lenguaje de consulta creado por IBM que facilita el tratamiento de datos de cualquier base de datos relacional.

## 8. Bibliografía

### 8.1. Libros

- Thinking in Java.
- Java Data Objects
- "Java 2 - Manual de Programacion"

### 8.2. Direcciones web

<http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html>

<http://javaisjava.blogspot.com/2009/05/hacia-un-framework-de-persistencia.html>

<http://xxito.wordpress.com/2007/06/02/07-beans-parte-i/>

<http://es.wikipedia.org/wiki/Wikipedia:Portada>

<http://www.mysql.com/>

<http://www.chuidiang.com/java/mysql/EjemploJava.php>

<http://www.chuidiang.com/java/herramientas/maven.php>

<http://ldc.usb.ve/~mgoncalves/IS3/Clase%201c%20IS-Persistencia.pdf>

<http://riunet.upv.es/handle/10251/8922>

<http://mikiorbe.wordpress.com/2009/06/24/descripcion-de-ibatis-%C2%BFque-es-%C2%BFpara-que-sirve/>

<http://www.hibernate.org/>

<http://www.oracle.com/technetwork/java/index-jsp-135919.html>

<http://www.proactiva-calidad.com/java/ejb/introduccion.html>

## **I. Anexo**

Se ha generado la documentación javadoc para que junto a los fuentes entregados sea más comprensible el funcionamiento interno del framework.