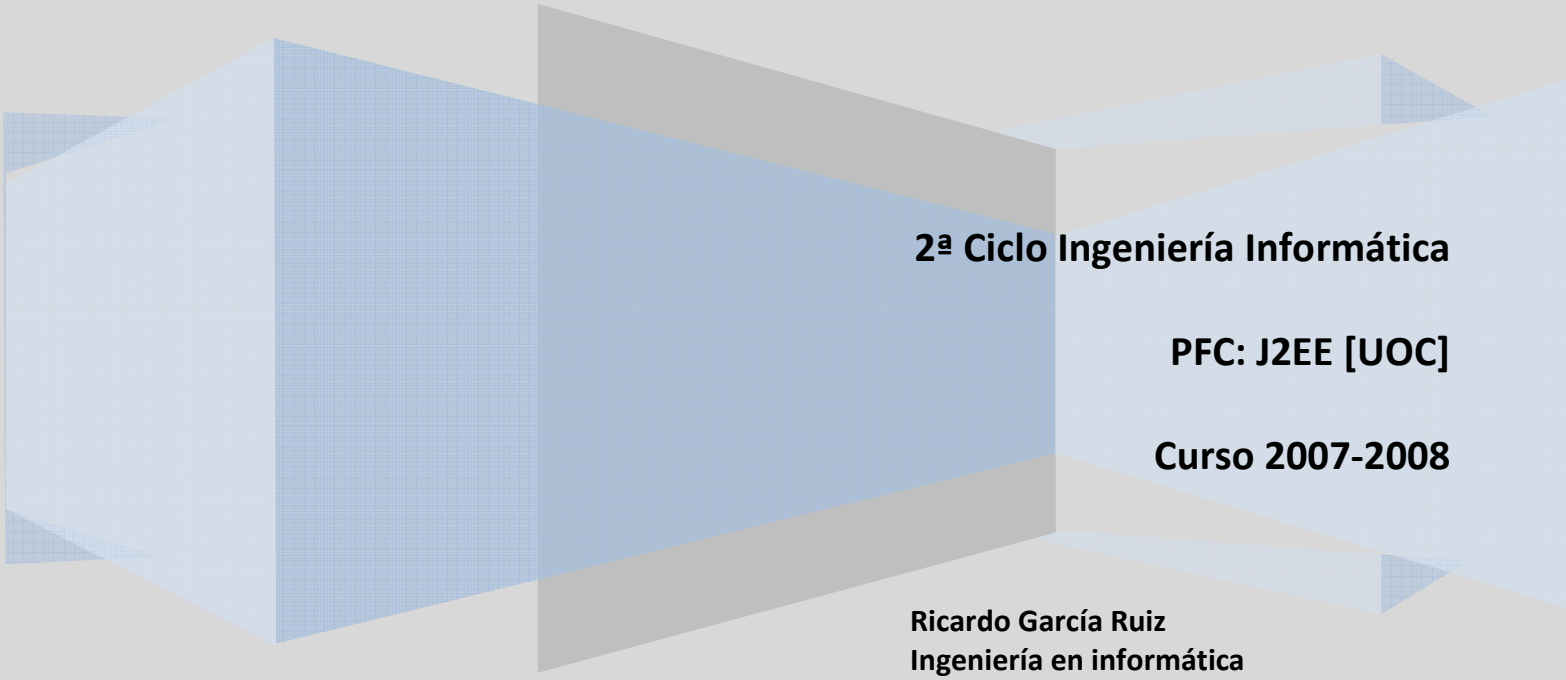
PFC: J2EE

Diseño e Implementación de un Framework de Persistencia



2ª Ciclo Ingeniería Informática

PFC: J2EE [UOC]

Curso 2007-2008

Ricardo García Ruiz
Ingeniería en informática
Consultor: Josep María Camps Riva

INDICE

Descripción del PFC

Los frameworks orientados al objeto (simplemente frameworks) son la piedra angular de la moderna ingeniería del software. El desarrollo del framework está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente (*source code*). Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados.

Dentro de la múltiple variedad de posibilidades de creación y utilización de frameworks, en este PFC nos vamos a centrar en frameworks creados bajo la plataforma Java y orientados a persistencia de datos.

¿Qué es la persistencia de datos? "*A persistence framework moves the program data in its most natural form (in memory objects) to and from a permanent data store the database. The persistence framework manages the database and the mapping between the database and the objects. There are many persistence frameworks (both Open Source and Commercial) in the market. Persistence framework simplifies the development process.*"

Es decir, el framework de persistencia se encarga de gestionar completa y transparentemente la lógica de acceso a los datos en un SGBD (Sistema de Gestión de Bases de Datos), ya sea de entrada o salida, y ocultando los detalles más pesados relativos a la estructura propia de la Base de Datos utilizada.

No obstante lo anterior, el objetivo de este PFC es realizar un estudio completo desde el ámbito de lo general (qué es un framework), pasando por el análisis de las características de los framework, hasta llegar al detalle en lo referente a los framework de persistencia.

Abstract

The frameworks orientated to the object (simply frameworks) are the angular stone of modern engineering of software. The development of the framework is gaining rapidly the acceptance due to your aptitude to promote the reutilization of the code of the design and the source code. Frameworks are the Generators of Application that relate directly to a specific domain), that is to say, with a family of related problems.

Inside the multiple varieties of possibilities of creation and utilization of frameworks, this PFC will centre in frameworks created under Java's platform and orientated to persistence of information.

What is the persistence of information? "*A persistence framework moves the program data in its most natural form (in memory objects) to and from a permanent data store the database. The persistence framework manages the database and the mapping between the database and the objects. There are many persistence frameworks (both Open Source and Commercial) in the market. Persistence framework simplifies the development process.*"

That is to say, persistence's framework takes charge managing completes and transparently the logic of access to the information in a SGBD (System of Management of Databases), so much of entry as exit, and concealing the most difficult details relative to the structure of used Database.

Nevertheless, the aim of this PFC is to realize a complete study, beginning for the most general thing (what is a framework?), continuing to the analysis of the characteristics of the framework, and finishing coming to the detail in what concerns the framework of persistence.

Resum

Els frameworks orientats a l'objecte (simplement frameworks) són la pedra angular de la moderna enginyeria del programari. El desenvolupament del framework està guanyant ràpidament l'acceptació a causa de la seva capacitat per a promoure la reutilització del codi del disseny i el codi font (*source code*). Els frameworks són els Generadors d'Aplicació que es relacionen directament amb un domini específic, és a dir, amb una família de problemes relacionats.

Dintre de la múltiple varietat de possibilitats de creació i utilització de frameworks, en aquest PFC ens anem a centrar en frameworks creats sota la plataforma Java i orientats a persistència de dades.

Què és la persistència de dades? "*A persistence framework moves the program data in its most natural form (in memory objects) to and from a permanent data store the database. The persistence framework manages the database and the mapping between the database and the objects. There are many persistence frameworks (both Open Source and Commercial) in the market. Persistence framework simplifies the development process.*"

És a dir, el framework de persistència s'encarrega de gestionar completa i transparentem-ne la lògica d'accés a les dades en un SGBD (Sistema de Gestió de Bases de dades), ja sigui d'entrada o sortida, i ocultant els detalls més pesats relatius a l'estructura pròpia de la Base de dades utilitzada.

No obstant això l'anterior, l'objectiu d'aquest PFC és realitzar un estudi complet des de l'àmbit del general (quin és un framework), passant per l'anàlisi de les característiques dels framework, fins a arribar al detall referent als framework de persistència.

Índice de Figuras

Índice de Tablas

1 Introducción

1.1 Objetivos específicos del PFC

El proyecto de desarrollo del PFC pretende abarcar en la medida de lo posible los siguientes aspectos:

- Definición de framework.
- Características generales de un framework.
- Tipos de framework existentes.
- ¿Qué es la persistencia de datos?: características.
- Frameworks específicos para la persistencia de datos.
- Búsqueda de información sobre frameworks open source y propietarios relacionados con la persistencia de datos.
- Selección de los frameworks más representativos y realización de un análisis comparativo de rendimiento e implantación de los mismos.
- Diseño de framework de persistencia
    - Análisis de requisitos de un framework de persistencia.
    - Características a observar en el diseño:
        - Análisis del dominio
        - Diseño del framework
        - "instanciación" del framework
- Implementación del framework de persistencia usando tecnología java.
- Desarrollo de una aplicación simple de ejemplo para verificar su funcionamiento en real.

1.2 Características específicas del PFC

El desarrollo de una aplicación framework de persistencia es muy complejo, por lo que se intentará desarrollar la implementación específica de la manera más simple posible, y reduciendo el campo de aplicación a algún sistema de Gestión de Bases de Datos generalista (MySQL, Oracle…), indicándose en la Memoria la valoración de su uso final.

La utilización de esta tecnología, con formatos tan variados como lo son los sistemas distribuidos y además los sistemas de Bases de Datos, requiere de una concreción en la tipología de trabajo y en las características de los elementos que se emplearán en su desarrollo:

- La aplicación se desarrollará en Java para poder ser utilizada en plataformas diversas (MS Windows 98, ME, 2000, XP, Vista y sistemas operativos Linux.
- Las versiones estándar que se usarán para el desarrollo y planificación, así como para las pruebas finales, serán:
    - MagicDraw, Personal Edition, versión 12.0, licencia Educación.
    - MagicDraw, Enterprise Edition, versión 15.0.
    - Las Bases de Datos pensadas originalmente para soportar el desarrollo de la implementación y el ejemplo final son, en principio, MySQL y Oracle, pero se tomarán en consideración otros sistemas de gestión de la persistencia en función del curso del desarrollo y la implementación final.
- En cuanto al desarrollo en la etapa del Diseño, se intentará facilitar la mayor cantidad de Diagramas UML posible, con el objeto de valorar y sistematizar el diseño a fin de que pueda ser tomado como base para futuras ampliación del modelo. Concretamente, como mínimo:
    - Modelos estáticos: diagramas de clases.
    - Diagramas de secuencia.
    - Diagramas de uso (si es útil finalmente).
    - La tecnología UML será la 2.0 o superior.

1.3 Características de la aplicación a desarrollar

Este PFC va encaminado al análisis, diseño e implementación de componentes de software, típicamente llamado framework, con la idea de servir de base para desarrollos sencillos directos así como para rediseños escalares posteriores.

En este sentido, la aplicación debería de reunir, al menos, las siguientes características:

- Uso de tecnología java, para su utilización en diversas plataformas. Típicamente se usará como base la versión 5.0 o superior.
- Separación clara entre el software de análisis y conversión y el de representación.
- Se potenciará el uso de patrones de diseño, especialmente el MCV (Modelo Vista Controlador), ya que la capa de persistencia está orientada en mejor modo, desde un punto de vista de las aplicaciones distribuidas, bajo esta paradigma de diseño ampliamente utilizado. De esta manera el diseño del framework podría ser utilizado con mejor desempeño en desarrollos J2EE que usen este modelo MVC.
- En el análisis y desarrollo se potenciará el uso de tecnología de libre acceso. En el caso de tener que usar tecnología propietaria se deberá de justificar su uso.
- El desarrollo del software, framework, debe realizarse de forma que pueda ser ampliable y reutilizable en la medida de lo posible. Es decir, nos referimos no solo a que pueda ser reutilizado y ampliado por terceros con sencillez, sino también a que puedan ser ampliadas las funcionalidades de persistencia, de forma que esto no suponga en ningún caso una reescritura del código previamente definido.
- Se valorará la posibilidad de que pueda ser gestionado el framework dentro de alguna plataforma IDE de uso común (Eclipse, Netbeans,…), aunque está función no es prioritaria en modo alguno.

1.4 Cronología de desarrollo del PFC

1. Tareas correspondientes a la fase de toma de datos y análisis de frameworks:

- Comprensión de framework:
  - Búsqueda de información en Internet.
  - Análisis de los modelos de framework.
  - Investigación sobre modelos open source y propietarios.
- Análisis de las herramientas generales framework:
  - Elementos generales del uso en framework.
  - Estructura de framework.
  - Diseño general de un framework
- Análisis de la persistencia de datos.
  - Características de la persistencia.
  - Tipos de sistemas de gestión de la persistencia.
  - Tecnología para gestión de la persistencia.
- Búsqueda de información sobre modelos de frameworks orientados a la gestión de la persistencia:
  - Modelos open source
  - Modelos propietarios
  - Comparación de los modelos más significativos

2. Tareas relacionadas con el diseño de la aplicación framework de persistencia:

   El desarrollo de la aplicación seguirá las fases del ciclo de vida en cascada de forma iterativa e incremental:

   - Recogida y documentación de los requisitos para el framework:
     - Elaboración de los diagramas necesarios para la comprensión y sistematización del modelo, esto es diagrama de casos de uso.
     - Documentación textual de los casos de uso.
   - Análisis:
     - Revisión de los casos de uso de la etapa previa.
     - Identificación de las clases. Representación mediante diagramas de colaboración para cada uno de los casos de uso.
     - Identificación de las relaciones entre las clases de entidad y elaboración de un diagrama estático UML.
     - Especificación formal de los casos de uso.
     - Especificación de los diversos diagramas UML necesarios.
   - Diseño:
     - Explorar la posibilidad de utilizar patrones adicionales.
     - Establecimiento de los subsistemas de la aplicación, si son necesarios.
     - Especificación de las pruebas que describan con que datos se ha de probar cada programa o grupo de programas pertenecientes a un módulo y cuáles son los resultados esperados en cada caso.
     - Interacción con los Sistemas de Gestión de Bases de Datos.
   - Implementación o codificación para cada uno de los módulos:
     - Implementación de las clases del framework utilizando el lenguaje java.
   - Prueba de la aplicación y test de resultados:

- Diseño y especificación de un plan de pruebas, tomando las especificaciones de las pruebas elaborados durante la etapa de diseño, así como la validación de los requisitos.
- Diseño de una aplicación de ejemplo.

3. Tareas relativas a la instalación de software.
   - Entrega de la aplicación con sus módulos software de código, clases java, y si es posible contenedor jar.
   - (Opcional) Integración del framework en el IDE seleccionado.
4. Confección de la documentación final del proyecto
   - Elaboración de un manual de usuario para el uso del framework de persistencia.
   - Revisión de la documentación interna del software (documentación de las clases).
   - Elaboración de un javadoc.
5. Elaboración de la memoria del PFC
6. Elaboración de la presentación del PFC

1.5 Temporización

ID	TAREA	Duración	Comienzo	Fin	Predecesoras
1	Toma de datos y análisis de Frameworks	28,d	10/03/2008	13/04/2008	
1.1	Datos generales frameworks	5,d	10/03/2008	14/03/2008	
1.2	Análisis herramientas framework	10,d	15/03/2008	26/03/2008	1.1
1.3	Análisis de la persistencia de Datos	5,d	27/03/2008	02/04/2008	1.2
1.4	Frameworks de Persistencia	7,d	04/04/2008	13/04/2008	1.3
2	PEC2	24,d	14/03/2008	13/04/2008	1
3	Entrega PEC2	1,d	14/04/2008	14/04/2008	2
4	Diseño del Framework	57,d	07/04/2008	19/06/2008	
4.1	Recogida y documentación de los requisitos para el framework	10,d	07/04/2008	17/04/2008	
4.2	Análisis	15,d	12/04/2008	30/04/2008	4.1
4.3	Diseño	20,d	21/04/2008	16/05/2008	4.2
4.4	Implementación o codificación para cada uno de los módulos	30,d	05/05/2008	13/06/2008	4.3
4.5	Prueba de la aplicación y test de resultados	20,d	26/05/2008	19/06/2008	4.4
5	PEC3	26,d	14/04/2008	16/05/2008	2
6	Entrega PEC3	1,d	16/05/2008	16/05/2008	5
7	Confección de la documentación del proyecto	40,d	28/04/2008	19/06/2008	4
8	Elaboración Memoria PFC	71,d	18/03/2008	19/06/2008	01-jul
9	Elaboración presentación PFC	8,d	14/06/2008	24/06/2008	
10	Entrega final	1,d	25/06/2008	25/06/2008	8, 9

Tabla 1: Planificación de las tareas de resolución del PFC

Contenido de las entregas parciales:

PEC2:

Entrega de la memoria con los resultados de la toma de datos, incluyendo parcialmente los modelos de traslación, que ya estarán avanzados.

También parte de la documentación de requisitos

PEC3:

Entrega de la memoria y de la aplicación con los pasos de la toma de Datos y el Diseño de la aplicación (de forma no definitiva, a falta de la evaluación de los test de funcionamiento, y sin el manual y la documentación).

Entrega final:

Entrega de la memoria.

Entrega de la aplicación y su documentación (javadoc y manual).

Entrega de la presentación.

Evaluación del material necesario

Se prevé que la principal fuente de información para el desarrollo de la aplicación sea Internet. Así mismo, todo el software necesario para la implementación de la aplicación (Linux, MagicDraw, IDE Eclipse, Netbeans 5.0) es libre o en versión educación, pudiéndose descargar desde sus respectivos sitios web.

En el caso de MagicDraw Enterprise Edition, versión 15.0, se ha evaluado su calidad superior a la versión 12.0, y es factible que se use en la versión de pago.

Riesgos y planes de contingencia

Los posibles riesgos vendrán derivados de las siguientes incidencias:

- Dificultades técnicas.
- Dificultad de utilización de las herramientas de software en la etapa de implementación.
- Curva de aprendizaje de los *patrones* en caso de utilizarlos
- Imprevistos de índole personal-familiar.
- Aumentos de la carga y horario laboral.

En previsión a los riesgos relacionados con la etapa de implementación, el plan de contingencia consiste en poder destinar un mayor plazo de ejecución para dicha etapa en contraste con el resto de etapas (ver Temporización).

Para el resto de riesgos no previsibles, el plan de contingencia principal consiste en ampliar los plazos, disponiendo de los días de reserva. En ausencia de incidencias, dichos días serán destinados al repaso de la presentación y de la memoria.

1.6 Diagrama de Gantt

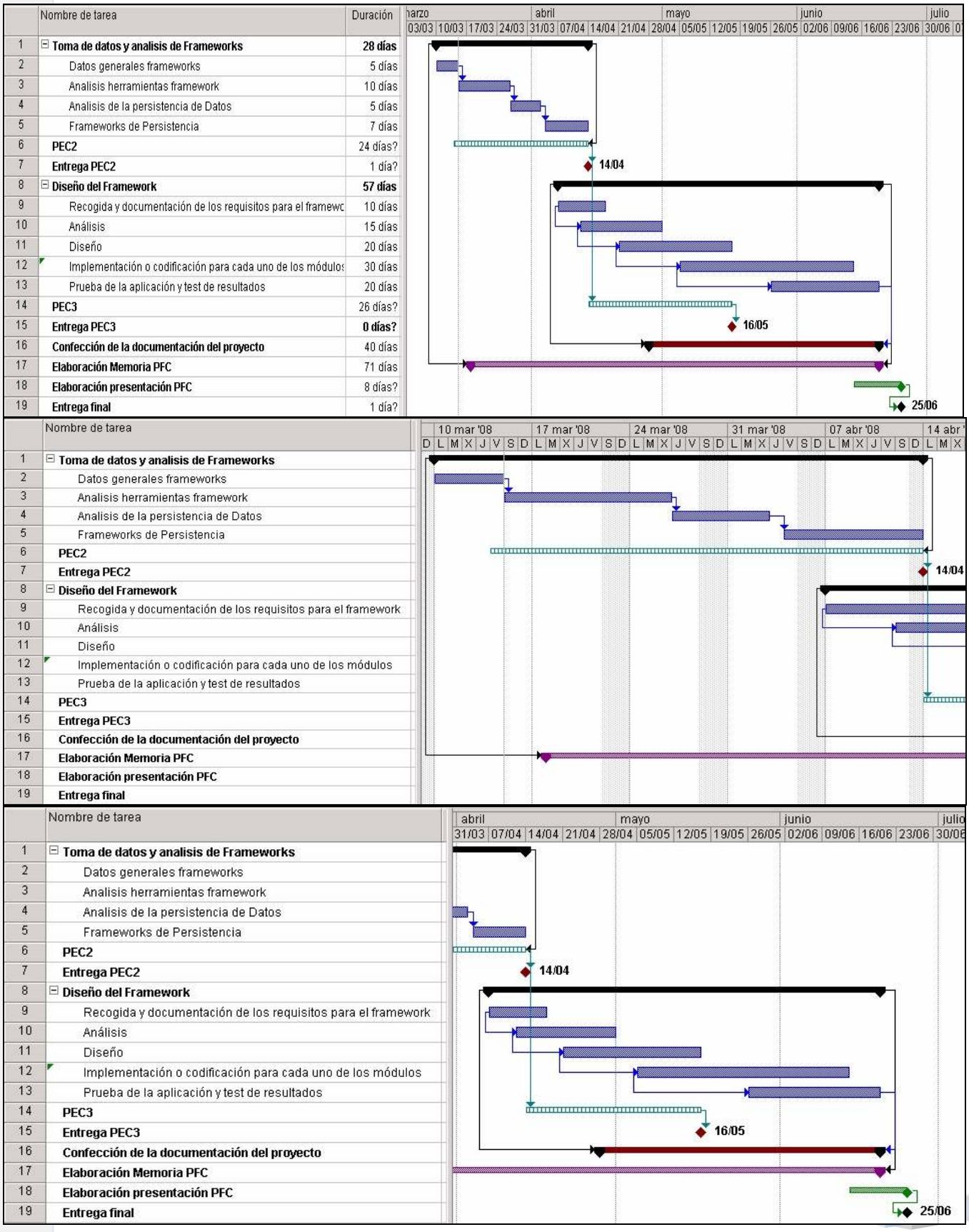


Figura 1: Diagrama de Gantt, planificación del PFC

2 Características inherentes a un framework

2.1 Definición de framework

Dentro del ámbito de la programación, un framework es un conjunto de funciones o código genérico que para realizar tareas comunes y frecuentes en todo tipo de aplicaciones (creación de objetos, conexión a base de datos, limpieza de strings, etc.). Esto da la oportunidad para mantener una sólida base sobre la cual desarrollar aplicaciones específicas, permitiendo obviar los componentes más triviales y genéricos del desarrollo.

Básicamente, los frameworks son construidos en base a lenguajes orientados a objetos. Esto permite la modularización 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.

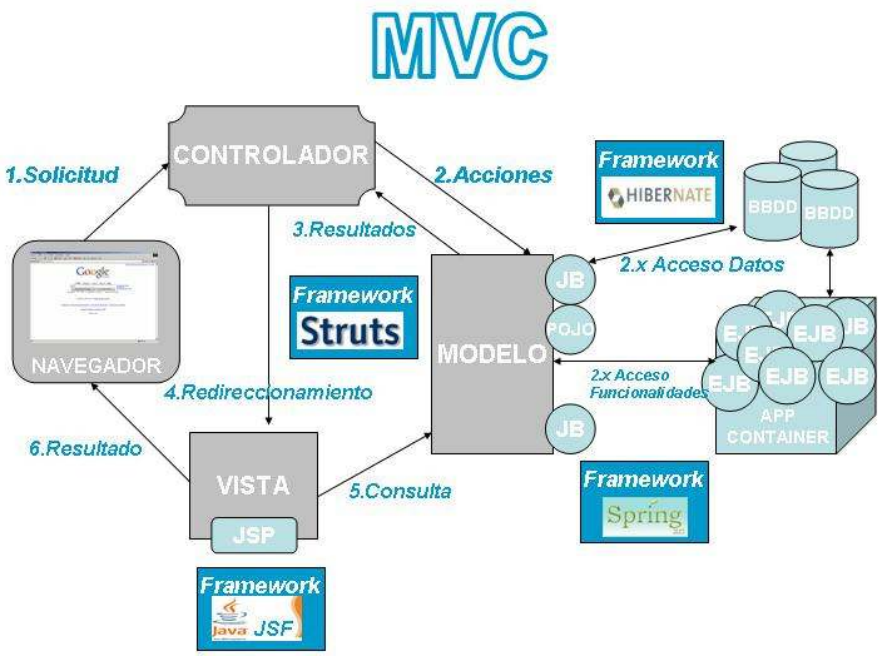


Figura 2: Esquema MVC, Modelo - Vista - Controlador

Dentro del proceso de desarrollo de software, un Framework consiste en una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Un Framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Provee de una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

En general, con el término Framework, nos estamos refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un Framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un Framework son:

- acelerar el proceso de desarrollo
- reutilizar código ya existente
- promover buenas prácticas de desarrollo como el uso de patrones

2.2 El patrón MVC

El patrón Modelo-Vista-Controlador es una guía para el diseño de arquitecturas de aplicaciones que ofrezcan una fuerte interactividad con usuarios.

El patrón MVC consiste en separar de la mejor forma posible las capas:

- Modelo (los objetos que interactúan con la base de datos y efectúan los procesos pesados o "lógica de negocios"), representa los datos de la aplicación y sus reglas de negocio
- Vista (la presentación final de los datos procesados al cliente, comúnmente en formato HTML), conjunto de vistas que representa los formularios de entrada y salida de información , y
- Controlador (la capa que se encarga de recibir el input del usuario, delegar el trabajo a los Modelos apropiados e invocar las Vistas que correspondan), conjunto de controladores que procesa las peticiones de los usuarios y controla el flujo de ejecución del sistema

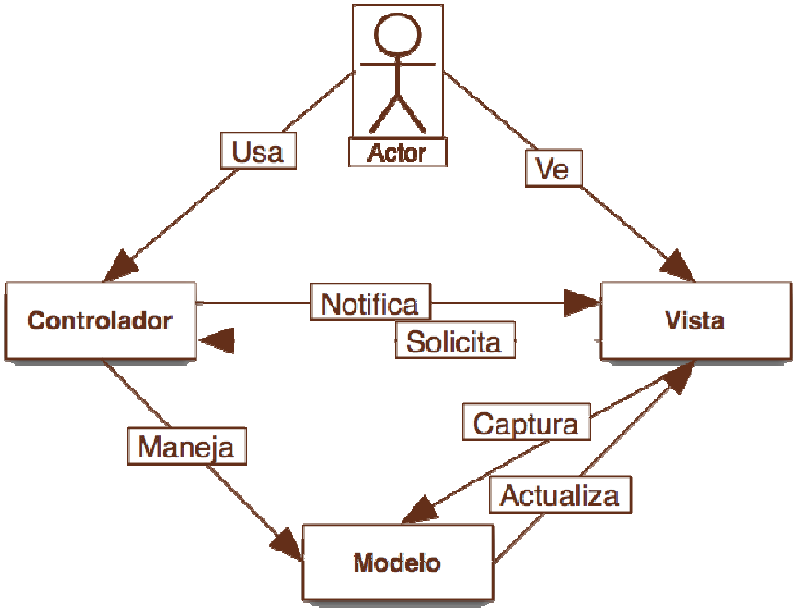


Figura 3: MVC, diagrama interacción usuario

2.3 Uso de patrón MVC y framework

De manera sintética se puede resumir en:

- El framework "llama" al código de nuestra aplicación.
- El desarrollador tiene que implementar interfaces y/o extender clases abstractas que proporcionará el framework

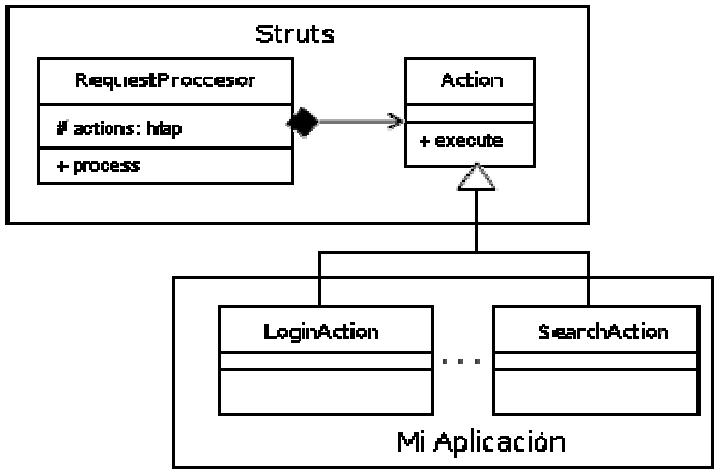


Figura 4: Ejemplo aplicación MVC en Struts

Así, cada desarrollo de un framework realiza una implementación determinada y específica del patrón MVC adaptándolo a la casuística determinada por la arquitectura a la que se orienta el framework, el dominio que se desea implementar y las características de despliegue del mismo.

2.4 La arquitectura J2EE

Aunque un framework, por definición, no está asociado a un tipo de plataforma de desarrollo, parece conveniente incluir (ya por que es la tecnología de uso en este TFC bien porque es la arquitectura de uso mayoritario en este momento) una mención al conjunto de desarrollos en Java de la empresa Sun.

Actualmente hay otros frameworks (tanto de orientación web como hacia la persistencia de datos) que son implementación en tecnologías diversas: .NET, PHP, Ruby on Rail,…

Sin embargo, J2EE reúne ciertas características que la hacen más proclive a su uso en este tipo de desarrollos:

- Lenguaje JAVA ya conocido
- Tecnología potente y madura
- Cubre la mayoría de necesidades tecnológicas: páginas dinámicas JSP, lógica de negocio mediante JAVA,…
- Interoperable con otras tecnologías: XML, JavaScript, HTML, …
- Gran cantidad de soporte en la red: APIs, manuales…
- OpenSource y herramientas de desarrollo gratuitas (Eclipse)
- Muchas utilidades ya creadas y fáciles de integrar
- Fácil conectividad con Base de Datos: driver JDBC (MySQL)
- Existencia de Frameworks de desarrollo basados en MVC (Struts)

J2EE es una herramienta claramente enfocada al mundo empresarial, orientada a un tipo de desarrollo específico.

En resumen, J2EE define un estándar para el desarrollo de aplicaciones empresariales multicapa (diseñado por Sun Microsystems). Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proveyendo un completo conjunto de servicios a estos componentes, y manejando muchos de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

3 Estado del arte de los Sistemas de Gestión de Bases de Datos (SGBD)

3.1 Introducción

El procesamiento de la información ha realizado una evolución considerable en los últimas años: pasando de los tiempos en que era realizado por grandes mainframes (tanto programas como datos eran almacenados en tarjetas perforadas) hasta la realidad actual de sistemas distribuidos, interoperables, con fuentes de datos heterogéneas y con capacidades de almacenamiento masivo.

Cualquier aplicación que requiera del procesamiento de datos puede se puede segmentar a grandes rasgos en dos niveles: datos persistentes y el procesamiento de los mismos.

Los mecanismos utilizados para interactuar con los datos son de vital importancia, tanto por su impacto en el desempeño final del sistema como también por los aspectos de mantenibilidad, diseño, reusabilidad y escalabilidad.

Como primera solución destacable para la persistencia de datos surgen las bases de datos.

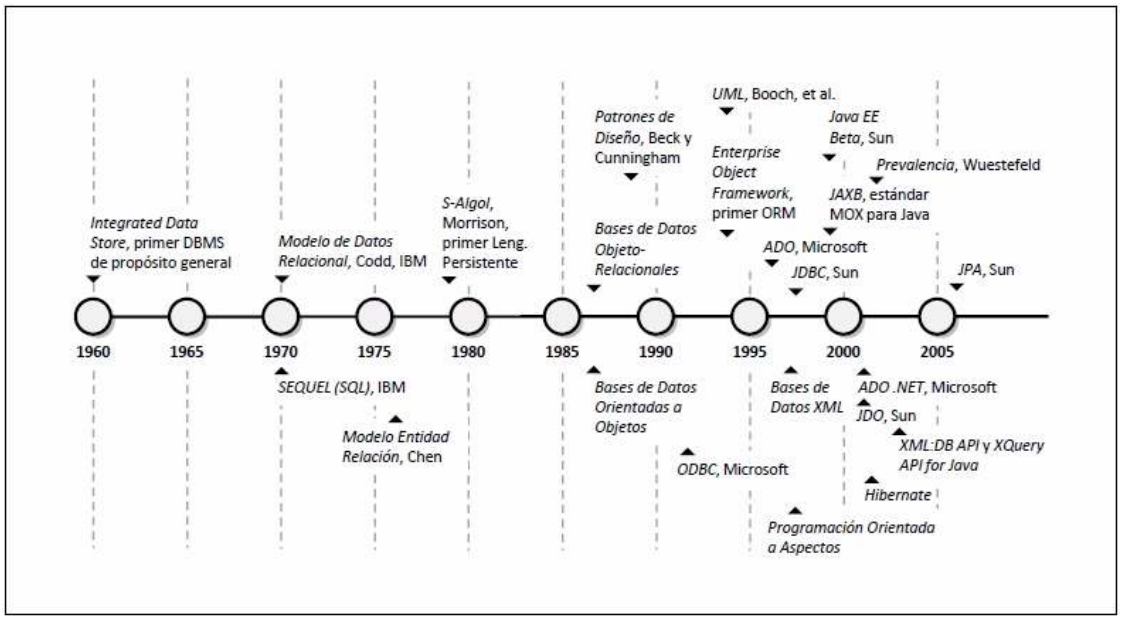


Figura 5: Esquema de desarrollo temporal de los SGBD

En los comienzos, el acceso a la información en estos SGBD constituía una tarea de muy bajo nivel.

Para manejar la creciente complejidad de los sistemas se tuvo que ir en la dirección de profundizar en unos mayores niveles de abstracción. Así surgen interfaces de acceso a datos y lenguajes de consulta estándares. Debido al auge de los lenguajes de programación orientados a objetos y la gran aceptación de las bases de datos relacionales, aparecen los denominados mapeadores como mecanismos de traducción entre los paradigmas subyacentes. A la par del avance tecnológico, los procesos de desarrollo de software evolucionaron integrando nuevas herramientas de abstracción como ser lenguajes de modelado. De igual manera, la experiencia de la comunidad de desarrolladores dio a conocer buenas soluciones para problemas conocidos, a las cuales se les denominaron patrones de diseño. Estos se beneficiaron de los lenguajes de modelado como herramienta para especificar soluciones. Tomando en cuenta que un sistema a menudo requiere la aplicación repetitiva de distintos patrones, se popularizan herramientas de generación automática de código.

Aunque en la actualidad un conjunto de tecnologías para persistencia lideran el mercado, sí existen alternativas de menor difusión pero con enfoques diferentes: lenguajes persistentes, orientación a aspectos o prevalencia.

3.2 Evolución de los SGBD

Para almacenar grandes volúmenes de datos, controlar el acceso concurrente a los mismos y asegurar la consistencia y seguridad de la información surge el concepto de Database Management System (SGBD), SGBD – Sistemas de Gestión de Bases de Datos.

Puede definirse un SGBD: paquete o sistema de software diseñado para facilitar la creación y el mantenimiento de bases de datos computarizadas. Asimismo, se entiende Base de Datos como una colección de datos relacionados.

El primer SGBD de propósito general fue Integrated Data Store, diseñado por Charles Bachean en General Electric a principios de la década del 60. Fue la base para el Modelo de Datos de Red e influenció los sistemas de base de datos en esta época. A finales de los 60, IBM presenta el Information Management System (IMS), aun hoy en uso. El IMS constituyó la base para un método de representación de datos alternativo: ***Modelo de Datos Jerárquico***.

Las bases de datos pre-relacionales no contaban con un conjunto de comandos que permitiese trabajar con los datos. Cada base tenía su propio lenguaje o utilizaba programas escritos en COBOL o C para manipular registros. Asimismo, eran virtualmente inflexibles y no permitían ningún cambio estructural sin tener que detener su ejecución y reescribir gran cantidad de código. Esto último funcionó de forma relativamente efectiva hasta fines de los 60 donde las aplicaciones se encontraban basadas estrictamente en procesamientos por lotes. A principio de los años 70, el crecimiento de aplicaciones que requerían interacción por parte de los usuarios demandó la necesidad de algo más flexible. Situaciones en que un campo extra era requerido o un número de subcampos excedían la máxima cantidad permitida en determinado archivo se volvieron más y más comunes. IBM, en 1970, bajo la dirección de Edgar Codd, y trabajando en el San Jose Research Laboratory, propuso un nuevo modelo de datos: **Modelo de Datos Relacional**.

Este nuevo modelo produjo el rápido desarrollo de varios SGBDs basados en el mismo, así como importantes resultados teóricos que ubicaron el campo sobre una base firme. Los sistemas de base de datos maduraron como una disciplina académica, y la popularidad de los SGBDs relacionales se extendió hasta el plano comercial: *el uso de SGBDs para administrar información corporativa se convirtió en una práctica estándar*.

A partir de 1980, el Modelo de Datos Relacional se consolidó como el paradigma dominante, y los sistemas de base de datos continuaron ganando aceptación.

A finales de los 80 y principio de los 90, se lograron varios avances en el área de sistemas de base de datos. Se realizó una considerable investigación para obtener lenguajes de consulta más poderosos y modelos de datos más expresivos: gran énfasis en el análisis de datos complejos.

IBM, Oracle e Informix extendieron sus sistemas con la habilidad para almacenar nuevos tipos de datos (p.ej. imágenes) así como con la capacidad de realizar consultas más complejas. Sistemas especializados fueron desarrollados por varias empresas a fin de poder crear Data Warehouses, integrar información de distintas bases, y llevar a cabo análisis especializados de la información. Concretamente en este tiempo emergen los denominados SGBDs Orientados a Objetos (SGBDOO) y los SGBDs Objeto-Relacional (SGBDOR).

La primera generación de SGDBOOs data del año 1986 y su objetivo era el de proveer persistencia para lenguajes orientados a objetos (relacionados con la inteligencia artificial). Se trataban de sistemas standalone, se basaban en lenguajes propietarios y no hacían uso de plataformas estándar a nivel de la industria. El lanzamiento de Ontos en 1989 por la compañía del mismo nombre marcó el comienzo de la segunda generación en el desarrollo de SGDBOOs. Esta generación se basó en utilizar una arquitectura Cliente/Servidor y plataformas conocidas como C++, X-Window y UNIX. Itasca, primer producto de la tercera generación, fue lanzado al mercado en agosto de 1990. Si bien los SGDBOOs de primera generación pueden ser considerados como extensiones a lenguajes orientados a objetos los sistemas de tercera generación pueden ser definidos como SGBDs con características avanzadas y lenguajes orientados a objetos para la definición y manipulación de datos.

Un aspecto interesante a tener en cuenta es que si bien la orientación a objetos se ha establecido firmemente en el mercado como uno de los paradigmas de desarrollo dominantes, su popularidad no

tiene relación con la popularidad de los SGDBOOs. En concreto, la adopción de las bases de datos orientadas a objetos ha sido considerablemente limitada. Actualmente, los SGBD relacionales dominan el mercado: es poco probable que sean remplazados, en el corto plazo, por los SGDBOOs.

A mediados de los 90, con el auge de Internet y la popularidad de HTML como formato para desplegar información en este medio surge XML. Inicialmente desarrollado como estándar para la Web a fin de permitir que los datos estuviesen claramente separados de los detalles de presentación, rápidamente se notó que XML cubría la necesidad de una sintaxis flexible para intercambiar información entre aplicaciones.

XML no fue inicialmente desarrollado como un lenguaje para persistir información, pero al ser adoptado por numerosas organizaciones para llevar a cabo sus procesos de negocios, devino la necesidad de contar con un mecanismo para almacenar los datos en dicho formato.

Alternativas como persistencia de los documentos en el sistema de archivos o en una base relacional presentaron numerosas desventajas. Así surgen bases de datos capaces de entender la estructura de documentos XML, llevando a cabo consultas sobre los mismos de una manera eficiente: Base de Datos XML.

Actualmente se acepta la clasificación de las anteriores en dos categorías:

- Base de Datos XML-Native
- Base de Datos XML-Enabled

XML-Enabled presentan un modelo de datos interno distinto a XML (Modelo Relacional) y un componente de software para la traducción correspondiente.

De forma general, este módulo de traducción entre modelos no puede manejar todos los posibles documentos XML. En términos de lo anterior, reserva su funcionalidad para la subclase de documentos que cumplen con determinado esquema derivado a partir de los datos almacenados en la base.

Finalmente, a diferencia de las XML-Enabled, las bases de datos XML-Native son aquellas que utilizan el modelo de datos XML directamente.

Esto es, utilizan un conjunto de estructuras para almacenar documentos XML, en principio, arbitrarios.

4 Persistencia

4.1 ¿Qué es una capa de persistencia?

Persistencia es el hecho de guardar permanentemente la información generada por una aplicación en un sistema de almacenado, con el objetivo de qué se pueda usar más adelante.

Se pueden encontrar diferentes definiciones del término persistencia, según distintos puntos de vista y autores. Veamos dos que con más claridad y sencillez, concretan el concepto de persistencia de objetos.

La primera, más antigua, dice así: "*Es la capacidad del programador para conseguir que sus datos sobrevivan a la ejecución del proceso que los creo, de forma que puedan ser reutilizados en otro proceso. Cada objeto, independiente de su tipo, debería poder llegar a ser persistente sin traducción explicita. También, debería ser implícito que el usuario no tuviera que mover o copiar los datos expresamente para ser persistentes*".

Esta definición nos recuerda qué es tarea del programador, determinar cuándo y cómo una instancia pasa a ser persistente o deja de serlo, o cuando, debe ser nuevamente reconstruida; asimismo, que la transformación de un objeto en su imagen persistente y viceversa, debe ser transparente para el programador, sin su intervención; y que todos los tipos, clases, deberían tener la posibilidad de que sus instancias perduren.

Otra definición dice así: "*Persistencia es «la capacidad de un lenguaje de programación o entorno de desarrollo de programación para, almacenar y recuperar el estado de los objetos de forma que sobrevivan a los procesos que los manipulan".* Esta definición indica que el programador no debería preocuparse por el mecanismo interno que hace un objeto ser persistente, sea este mecanismo soportado por el propio lenguaje de programación usado, o por utilidades de programación para la persistencia, como librerías, framework o compiladores.

En definitiva, el programador debería disponer de algún medio para poder convertir el estado de un objeto, a una representación adecuada sobre un soporte de información, que permitirá con posterioridad revivir o reconstruir el objeto, logrando que como programadores, no haya que preocuparse de cómo esta operación es llevada a cabo.

Así, en vista de lo anterior, la persistencia de los objetos es, como su nombre indica, el almacenamiento de los objetos.

Este hecho permite que siempre se mantengan disponibles para la aplicación, para que esta los pueda consultar, modificar, combinar, etc. Es decir, gestionarlos. La persistencia de los objetos debe conservar dicho objeto en el estado que el usuario considere interesante para sus objetivos.

Un objeto en sí mismo se compone de atributos, métodos y relaciones con otros objetos. Los atributos podríamos decir que son el equivalente a los registros o campos de una base de datos. Cada uno define los aspectos que componen un objeto, en definitiva todos aquellos aspectos que creemos que definen el objeto dentro de la realidad y que nos resulta imprescindible definir en nuestro programa. Los atributos pueden ser de tipo primitivo, referenciado u otros objetos. Los métodos se pueden definir como funciones u operaciones que trabajan sobre los mismos atributos del objeto, o sobre relaciones con otros objetos. Las relaciones entre objetos pueden ser la herencia, de uso o asociación, parametrizada y la composición, entre otras.

Las bases de datos trabajan con registros. Los registros son las unidades básicas de trabajo de la información de las bases de datos relacionales. Estas unidades se clasifican en diferentes tipos, según la información: numéricos, caracteres, etc. Son los llamados tipos primitivos, y forman lo que denominamos modelo de datos.

Todos los lenguajes no orientados a objetos tienen un modelo de datos muy parecido a las bases de datos relacionales. Las variables son de tipo primitivo al igual que los registros de las bases de datos relacionales.

Una aplicación que trabaja con variables de tipos primitivos puede trabajar directamente y requiere de pocas adaptaciones con una base de datos relacional, pero como se puede entender fácilmente, una base de datos relacional cualquiera no puede almacenar un objeto a partir de registros. Puede llegar a tener un equivalente, a partir de los registros, parecido al objeto, pero nunca podrá almacenar todas sus propiedades, métodos y relaciones con otros objetos que lo definen, puesto que una base de datos relacional no está preparada. La base de datos relacional no tiene forma de almacenar de forma directa los métodos y las relaciones. Es preciso adaptar de alguna manera los modelos de datos.

El modelo de datos de objetos lo podemos describir cómo: *Una colección de conceptos bien definidos matemáticamente que ayudan a expresar las propiedades estáticas y dinámicas de una aplicación con un uso de datos intensivo*.

Conceptualmente, una aplicación quizás caracterizada por:

- **Propiedades estáticas**: entidades (u objetos), propiedades, (o atributos) de estas entidad, y relaciones entre estas entidades.
- **Propiedades dinámicas**: operaciones sobre entidades, sobre propiedades o relaciones entre operaciones.
- **Reglas de integridad** sobre entidades y las operaciones.

La diferencia de los modelos de datos hizo que los programadores, necesitados de encontrar la forma en que los objetos fuesen persistentes, adaptaran los modelos de datos de alguna manera. Se trataba que encontraran un modelo correspondiente o equivalente entre los dos modelos de datos: los lenguajes OO y las bases de datos.

Se desarrollaron varios sistemas que intentaban hacer una correspondencia entre los modelos de datos de las bases de datos relacionales y los objetos.

Pero también aparecieron ideas que solucionaban el problema de la persistencia de forma más directo y automatizado, un sistema que consiguiera la correspondencia entre los modelos de datos: las **capas de persistencia**.

4.2 Objetivos de una capa de persistencia

¿Qué características debe tener una capa de persistencia para poder llegar a ser un sistema consolidado?

Si tomamos como base a Scott W.Ambler una capa de persistencia robusta dirigida a bases de datos relacionales debe cumplir los siguientes requerimientos:

- **Diversificación de los mecanismos de persistencia:** Debe poder trabajar con varios sistemas almacenado de datos, no con uno sólo. Algunos podrían ser bases de datos, sistemas de ficheros, etc..
- **Encapsulación cumplida del mecanismo de persistencia:** El sistema sólo debe poder almacenar, eliminar y recibir los objetos. Del resto de operaciones se debe ocupar la capa de persistencia.
- **Acciones sobre múltiples objetos:** La capa debe consultar o eliminar múltiples objetos a la vez.
- **Transacciones:** Debe poder trabajar con acciones combinadas: consultar - borrar. También debe permitir parametrizar las transacciones. O sea realizar todas las operaciones propias del lenguaje SQL, en una base de datos.
- **Identificadores de los objetos:** Atributo numérico para cada objeto, que permita identificarlo respecto de otros totalmente iguales.

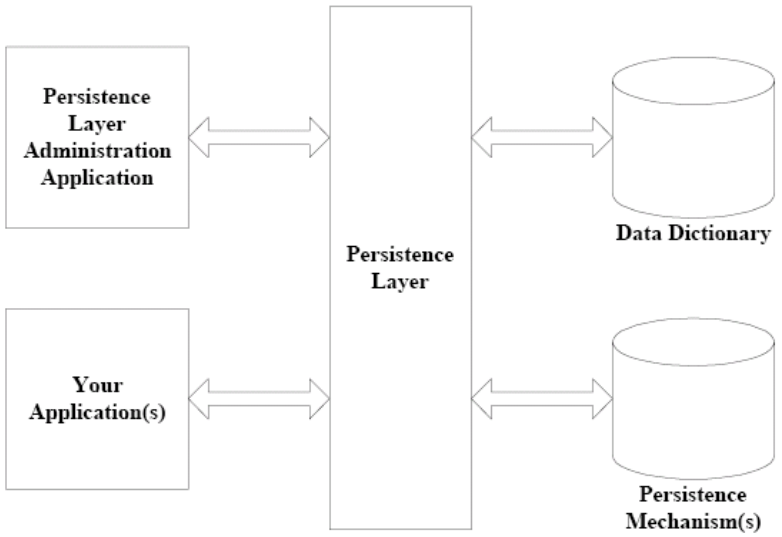


Figura 6: Esquema de interacción entre el SGBD, la capa Persistencia y las aplicaciones

- Cursores: De sistema para poder recibir los resultados de las operaciones de la base de datos.
- **Proxies:** Aproximación completaría a los cursores. Es un objeto representativo de otro pero sin entrar en el propio objeto, sino realizando un tratamiento superficial del objeto que representa. Es como un apuntador sobre los datos de un objeto.
- **Registros:** Para generar informes de resultados.
- **Múltiples arquitecturas:** Debe poder adaptarse tanto a arquitecturas cliente/servidor de dos capas como de múltiples capas.
- **Trabajar con diferentes bases de datos de diferentes fabricantes:** La capa de persistencia debe poder adaptarse fácilmente sin tener que modificar la aplicación.
- **Múltiples conexiones:** Debe poder trabajar con sistemas que no tienen la base de datos de forma centralizada: sistemas SGBD descentralizados.
- **Controladores nativos:** Debe trabajar con los sistemas de acceso a las bases de datos más comunes, o en su defecto, con los proporcionados por los fabricantes.

- **Consultas SQL.:** Debe permitir hacer consultas SQL, no de forma obligada, sino como una excepción, y nos casos muy especiales.

Estas características, bien fijadas e implementadas, deben poder conformar una capa de persistencia consistente y fiable, de forma que un programador experimentado pueda confiar y escogerla ante varias opciones.

Todas estas características conforman la denominada transparencia. Esta característica tiene como objetivo entre otros no restarle ninguna característica ni a las aplicaciones generadas por el lenguaje OO ni a las bases de datos, aprovechando al máximo todas las propiedades y características, en beneficio del conjunto.

La transparencia de la capa permite que tanto la aplicación como la base de datos trabajen como si se comunicaran entre ellas de forma directa, configurando un modelo de datos común. Incluso si cambiáramos la base de datos, la aplicación no tiene porque verse afectada en modo alguno, debiendo continuar trabajando del mismo modo. La transparencia se encarga de volver a conexionar los diferentes modelos de datos de forma automatizada, sin restar rendimiento y propiedades. Cómo podemos ver la transparencia respeta la independencia a las dos partes a la vez que permite que trabajen de forma conjunta como un sistema único. Este es el motivo por el que la persistencia por capas también se denomina persistencia transparente.

4.3 Los modelos de sistema de persistencia

La necesidad de la persistencia de los objetos ha conseguido que a lo largo del tiempo se hayan desarrollado diferentes sistemas a fin de conseguir este objetivo. Los sistemas más primitivos constituyeron la primera piedra en un proceso que ha permitido dar paso a sistemas más evolucionados, cómo son las capas de persistencia: el último peldaño evolutivo dentro de la escala de la persistencia de objetos. Casi todos de los diferentes sistemas de persistencia todavía se utilizan en mayor o menor grado.

Los principales sistemas de almacenamiento de datos con los que se trabaja habitualmente son las bases de datos. Entre las bases de datos se distinguen dos fundamentales: relacionales y orientadas a objetos. Además está la forma de bases de datos objeto-relacionales, mezcla de los dos sistemas de bases de datos. Otros sistemas con los que nos encontramos, que permiten realizar almacenado de datos, son los sistemas de ficheros, cómo pueden ser ficheros ASCII o los documentos XML.

Otros sistemas clave que también se deben mencionar por su creciente importancia son los que trabajan con arquitecturas de objetos distribuidos como EJB, y las bases de datos XML.

Dentro de la evolución de los sistemas de persistencia de objetos se ha trabajado sobre todos los diferentes sistemas de almacenamiento. Los primeros principalmente sobre ficheros, y base de datos relacionales. Los más evolucionados, que son las capas de persistencia, además permiten trabajar

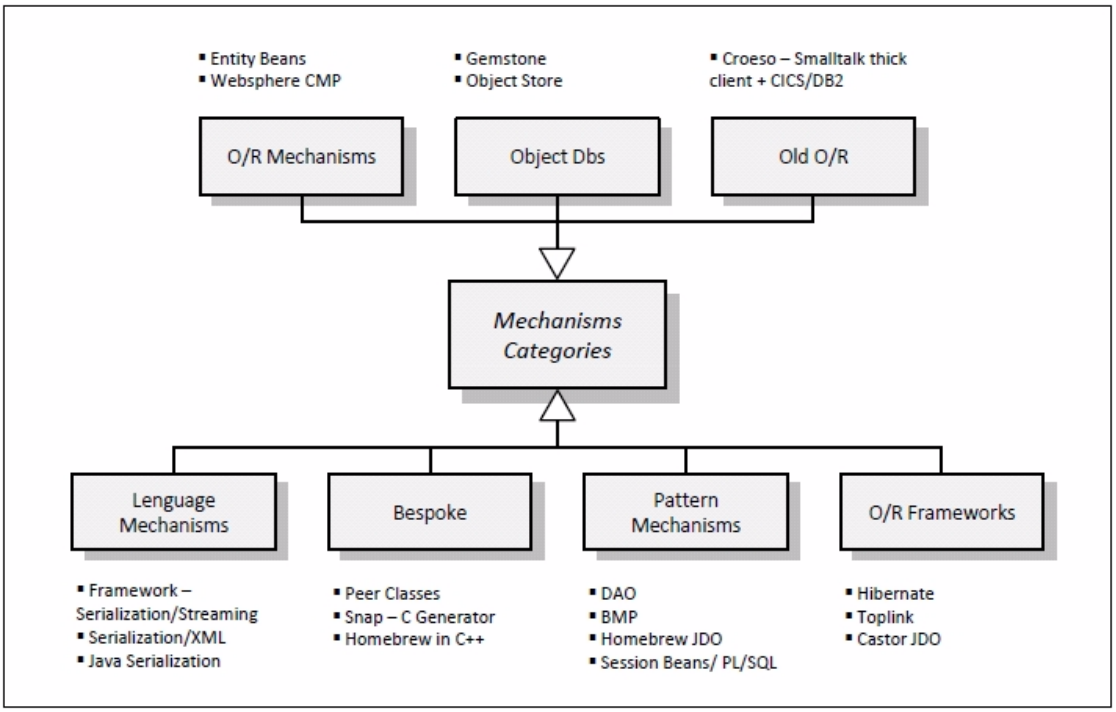


Figura 7: Agrupación o clasificación de los sistemas de persistencia

sobre sistemas distribuidos, y se empieza a hablar seriamente de trabajar sobre bases de datos XML.

No obstante, es preciso destacar que dentro de la diversidad de sistemas de persistencia, cómo podremos ver en la siguiente tabla comparativa, sólo las capas de persistencia están orientadas, tal y como indica Ambler, hacia múltiples sistemas de almacenado, ficheros, diferentes bases de datos, etc.

La siguiente tabla comparativa enumera algunos de los diferentes sistemas y capas de persistencia. Todos ellos son comparados según las características mencionadas por Ambler. Cómo podemos comprobar, los únicos sistemas de persistencia de objetos que cumplen la mayoría de las características son las capas de persistencia.

Requerimiento	Serialización	JDBC-ODBC	ODBMS	EJB	JDO	ODMG	ADO
Diferente tipos de mecanismos	Sistemas de archivos	SGBDR	SGBDOO	SGBDR, EAI, sistemas de archivos, etc.	Sistema de archivos SGBDR, SGBDOO, EAI, BBDD cobol, otros	Sistema de archivos SGBDR, SGBDOO, EAI, BBDD cobol, otros	SGBDR
Encapsulación completa del mecanismos de persistencia	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Acciones multiobjeto	No	Sí	Sí	Sí	Sí	Sí	Sí
Transacciones	No	Sí	Sí	Sí	Sí	Sí	Sí
Extensibilidad	No	No	No	No	Sí	Sí	Solo para tipo de objeto
Identificadores de los objetos	Sí, manual	Sí, manual	Sí	Sí	Sí	Sí	Sí
Cursores	No	Sí	No	Sí	Sí	Sí	Sí
Proxies	No		Sí	Sí	Sí	Sí	Sí
Registros	Sí, manual	Sí, manual	Sí	Sí	Sí	Sí	Sí
Soporte para múltiples arquitecturas	No	No	No	No	Sí	No	No
Posibilidad de trabajar con BBDD de diferentes fabricantes	No	Sí	No	Sí	Sí	Sí	Sí
Conexiones múltiples	No	No	Sí	Sí	Sí	No	No
Uso de controladores nativos o externos	No	No nativos	No	No nativos	No nativos	No nativos	No nativos
Consultas SQL	No	Sí	Sí	Sí	Sí	No	Sí

Tabla 2: Caracterización de los esquemas de persistencia

4.4 Conceptos de la persistencia de objetos

4.4.1 Instancia persistente y transitoria

Una instancia persistente es aquella cuyos datos perduran a la ejecución del proceso que materializó la instancia. Una instancia transitoria o temporal, es toda instancia cuyos datos desaparecen cuando finalizan los procesos que la manipulan. En ambos casos, las instancias en sí, como estructuras de datos residentes en memoria, desaparecen al finalizar los procesos que las crearon.

Un sencillo ejemplo: imaginemos la ejecución de un programa que solicita introducir nuestro nombre que será usado repetidas veces en distintas operaciones.

Si este dato es recogido en una instancia transitoria, cuando finalice el programa y lo volvamos a ejecutar, deberemos nuevamente introducir el dato; pero si está asociado a una instancia persistente, el dato introducido podría ser recuperado y mostrado en sucesivas ejecuciones del programa.

4.4.2 Servicio de persistencia de objetos

El concepto de servicio de persistencia es tema vigente de debate e investigación y desarrollo, en los mundos, académico y de la industria. Acotar con nitidez qué es un servicio de persistencia es una cuestión abierta, porque la separación entre este concepto y el de base de datos es difusa. Aquí, asumiendo que el destino final de los datos serán principalmente los sistemas de gestión de datos de la empresa, es adoptada la siguiente acepción en la mayor parte de los textos:

> *Servicio de persistencia es un sistema o mecanismo programado para posibilitar una interfaz única para el almacenamiento, recuperación, actualización y eliminación del estado de los objetos que pueden ser persistentes en uno o más sistemas gestores de datos.*

La definición anterior considera que el sistema gestor de datos puede ser un sistema SGBDR, un sistema SGBDOO, o cualquiera otro sistema; que el estado podría estar repartido sobre varios sistemas, incluso de distinto tipo; y lo más importante, que un servicio de persistencia de objetos aporta los elementos necesarios para efectuar la modificación y la eliminación de los objetos persistentes, además del volcado y recuperación del estado en los sistemas gestores de datos. Y todo ello, debería ser efectuado de acuerdo a la definición hecha más atrás de persistencia, sin necesidad de traducción explicita por parte del programador. En todos los casos, sea cual sea el tipo de gestor datos, los servicios de persistencia de objetos facilitan la ilusión de trabajar con un sistema de bases de datos de objetos integrado con el lenguaje de programación, ocultando las diferencias entre el modelo de objetos del lenguaje y el modelo de datos del sistema empleado como base de datos. A pesar de lo dicho, un servicio de persistencia no es un sistema de gestión de bases de datos orientado a objetos. El servicio de persistencia es un componente esencial de todo sistema gestor de bases de datos de objetos (SGBDOO), que resuelve otros aspectos, además de la persistencia.

De las alternativas estándar para hacer persistir los objetos en Java solo ODMG 3.0 y JDO pueden tener la consideración de servicio de persistencia de objetos Java. También es posible proponer utilizar un servicio de persistencia del OMG, PSS 2.0, para persistir objetos Java, pero este estándar considera opcionales las transacciones y la persistencia transparente, que son funciones necesarias para ofrecer un servicio de persistencia eficaz, conforme a las definiciones vistas de persistencia. JDBC, SQLJ requieren que el programador defina, e implemente las operaciones de persistencia teniendo en cuenta cómo convertir los objetos en tuplas. Ambos pueden ser utilizados en la construcción de servicios de persistencia para bases de datos relacionales. La serialización es el servicio de persistencia más básico, solo ofrece servicios para guardar y recuperar el estado de un objeto sobre ficheros y flujos de entrada salida.

De otra parte, los SGBD ofrecen servicios de persistencia, que adoptan una o varias de las siguientes aproximaciones de cómo se puede hacer un objeto persistente:

- ***Por tipo***. Un objeto puede llegar a ser persistente cuando es creado de algún tipo (clase) dotado de la capacidad de persistir o de un subtipo (clase descendiente) de estos. Los tipos persistentes son distintos de los tipos cuyas instancias son transitorias. El tipo podría ser identificado como persistente en su declaración, o por herencia de alguno de los tipos predeterminados por el sistema. De forma parecida, la serialización obliga que las clases implementen la interfaz Serializable.
- ***Por invocación explicita***. El programador invoca un método que provoca que un objeto pase a ser persistente. La llamada al método puede ser en la creación del objeto o en cualquier instante, según su implementación.
- ***Por referencia***. Un objeto es hecho persistente al ser referenciado por otro que es persistente. Añadir un objeto a una colección persistente, la asignación de un objeto a un atributo de otro persistente o la asignación a cierto tipo de referencias, provocan que un objeto transitorio pase a ser persistente.

4.4.3 Persistencia ortogonal

Dos características serán ortogonales si el uso de una no afecta a la otra, esto es, son independientes entre sí. Programas y persistencia serán ortogonales, si la forma en la que los objetos son manipulados por estos programas es independiente de la utilización de la persistencia, que los mismos mecanismos operaran tanto sobre objetos persistentes como sobre objetos transitorios, ambas categorías serían tratadas de la misma manera con independencia de su característica de persistencia. Ser persistente debería ser una característica intrínseca del objeto, soportada por la infraestructura del entorno de programación y persistencia. La persistencia de un objeto debe ser ortogonal al uso, tipo e identificación. Esto es, cualquier objeto debería poder existir el tiempo que sea preciso, ser manipulado sin tener en cuenta, si la duración de su vida, supera al proceso que lo creo, y su identificación no estar vinculada al sistema de tipos, como la posibilidad dar nombres a los objetos.

A la hora de plasmar el uso de la persistencia en nuestros programas, una persistencia ortogonal ideal, llevaría a no tener que modificar el código de nuestras clases, salvo aquellas donde debamos introducir las operaciones que provocan la persistencia para cualquier objeto que sea duradero. Los beneficios que aporta la persistencia ortogonal son importantes: mayores cotas de facilidad de mantenimiento, corrección, continuidad del código y productividad de desarrollo. Se consigue:

1. Menos código. Una semántica para expresar las operaciones de persistencia más simple de usar y entender. Evita la duplicidad de código uno preparado para instancias transitorias y otro para instancias persistentes.
2. Evitar la traducción explicita entre el estado de los objetos y su representación en base de datos, que redunda en mayor facilidad de mantenimiento y menos código también.
3. Facilitar la integridad y permitir que actúe el sistema de tipos subyacente, que automáticamente podría verificar la consistencia y la correspondencia de tipos, entre estados en base de datos y objetos en programa, la integridad no sería responsabilidad del programador.

Todo lo visto en este apartado apunta la conveniencia de usar persistencia ortogonal. En la práctica conseguir persistencia ortogonal completa no es fácil, habitualmente encontraremos limitaciones. Habrá clases de objetos que no son soportadas por los servicios de persistencia, bien por compromisos de diseño, como la dificultad de implementación; bien porque cabe pensar que determinados objetos no tienen sentido fuera del contexto de ejecución concreto de un proceso, como por ejemplo un puerto de comunicaciones para IP.

4.4.4 Cierre de persistencia

Los objetos suelen referenciar a otros objetos, estos a su vez a otros, y así puede continuar sucesivamente. Cada objeto puede tener un gran número de objetos dependientes de manera directa e indirecta. Esta relación de dependencias es parte integrante del estado de cada objeto. Cuando el estado de un objeto es salvado o recuperado, sus dependencias también deberían ser guardadas o recuperadas. De otro modo, cuando el objeto fuese recuperado, llegaría a estar incompleto, sería inconsistente con respecto a como fue guardado.

Un mecanismo de persistencia que posibilita la persistencia automática de las dependencias de un objeto, que deban persistir, se dice que admite el cierre de persistencia. Cuando el estado de un objeto es almacenado, los estados de los objetos dependientes que tengan que ser persistentes, son también almacenados, y así, sucesivamente. En otro sentido, en la recuperación del estado de un objeto, los estados de los objetos dependientes son recuperados. El cierre de persistencia determina el conjunto de referencias necesario, que ayuda a conseguir la consistencia entre el estado del objeto en el instante de guardar y estado resultante de su recuperación.

4.4.5 Persistencia por alcance

La persistencia por alcance o persistencia en profundidad es el proceso de convertir automáticamente en persistente todo objeto referenciado directa o indirectamente por un objeto persistente, los objetos del cierre de persistencia de un objeto son hechos persistentes. Es la aplicación recurrente de la estrategia de persistencia por referencia.

4.4.6 Transparencia de datos

Cuando un sistema o entorno de programación ofrece transparencia de datos, el conjunto de las clases persistentes y el esquema de la base de datos es uno, las clases definen de hecho el esquema en la base de datos. Los estados almacenados en la base de datos son manejados con el lenguaje de programación elegido, no es necesario otro. Los objetos son recuperados de la base de datos automáticamente, cuando las referencias a estos son accedidas. También, las modificaciones del estado de objetos persistentes son reflejadas en la base de datos automáticamente. Los estados de los objetos son recuperados y actualizados de forma transparente; no hay cambios en la semántica de referencia o de asignación en la aplicación. Objetos transitorios y persistentes son manipulados de igual forma. Las operaciones propias de la persistencia son efectuadas sin la intervención directa del programador, con más código. La frontera entre el lenguaje de programación y los servicios de datos desaparece a los ojos del programador, evitando la falta de correspondencia (*impedance mismatch*) entre la base de datos y lenguaje de programación.

No debemos confundir transparencia con persistencia ortogonal, la primera es una consecuencia de la segunda. Podemos encontrar transparencia de datos sin disponer de persistencia ortogonal total, es el caso de que determinadas clases no puedan persistir, esto concretamente supone que la solución no sería ortogonal, independiente, respecto el tipo.

4.4.7 Falta de correspondencia entre clases y datos

Cuando se trabaja con sistemas gestores de datos, como bases de datos relacionales, ficheros, bases de datos documentales XML, etc., cuyo modelo datos no tiene una equivalencia directa con el modelo de objetos del lenguaje de programación usado, hay una falta de correspondencia (*impedance mismatch*) entre la base de datos y lenguaje de programación, es necesario establecer un modelo de correspondencia, que defina como una clase se convierte en datos sobre el modelo ofrecido por sistema que albergará el estado de los objetos. A esta equivalencia entre la clase y los datos, se denomina aquí correspondencia clase - datos(object mapping). El caso particular de la correspondencia entre clases y tablas en un SGBDR, es la correspondencia objeto - registros. Se puede utilizar el término mapear para señalar al proceso de definición de la correspondencia entre las clases de los objetos persistentes y los modelos de datos, el proceso puede implicar cambios en ambos lados de la correspondencia, en el modelo de clases creado o modificado para asumir ciertos modelos de datos, y al contrario, el modelo de datos puede ser diseñado o cambiado, para permitir una correspondencia más eficiente y eficaz con ciertos modelos de clases.

Las definiciones de persistencia que se han visto invitan a emplear una persistencia que no cambie la forma de trabajar con el lenguaje de programación, que actúe de forma transparente y consistente, donde el modelo de clases y el modelo de datos son la misma cosa, y que sea una persistencia ortogonal. Los beneficios serán un código con menos líneas, más fácil de mantener, más correcto y productivo.

4.5 Mecanismos de persistencia

Como se puede observar se trata de una categorización en dos niveles que incluye tanto técnicas establecidas como ser acceso directo a bases de datos relacionales (una subcategoría de acceso directo a bases de datos) como también técnicas mas recientes y de creciente popularidad (p.ej. generadores de código y mapeadores objeto-relacional).

La existencia de un segundo nivel permite separar técnicas que mas allá de compartir una misma

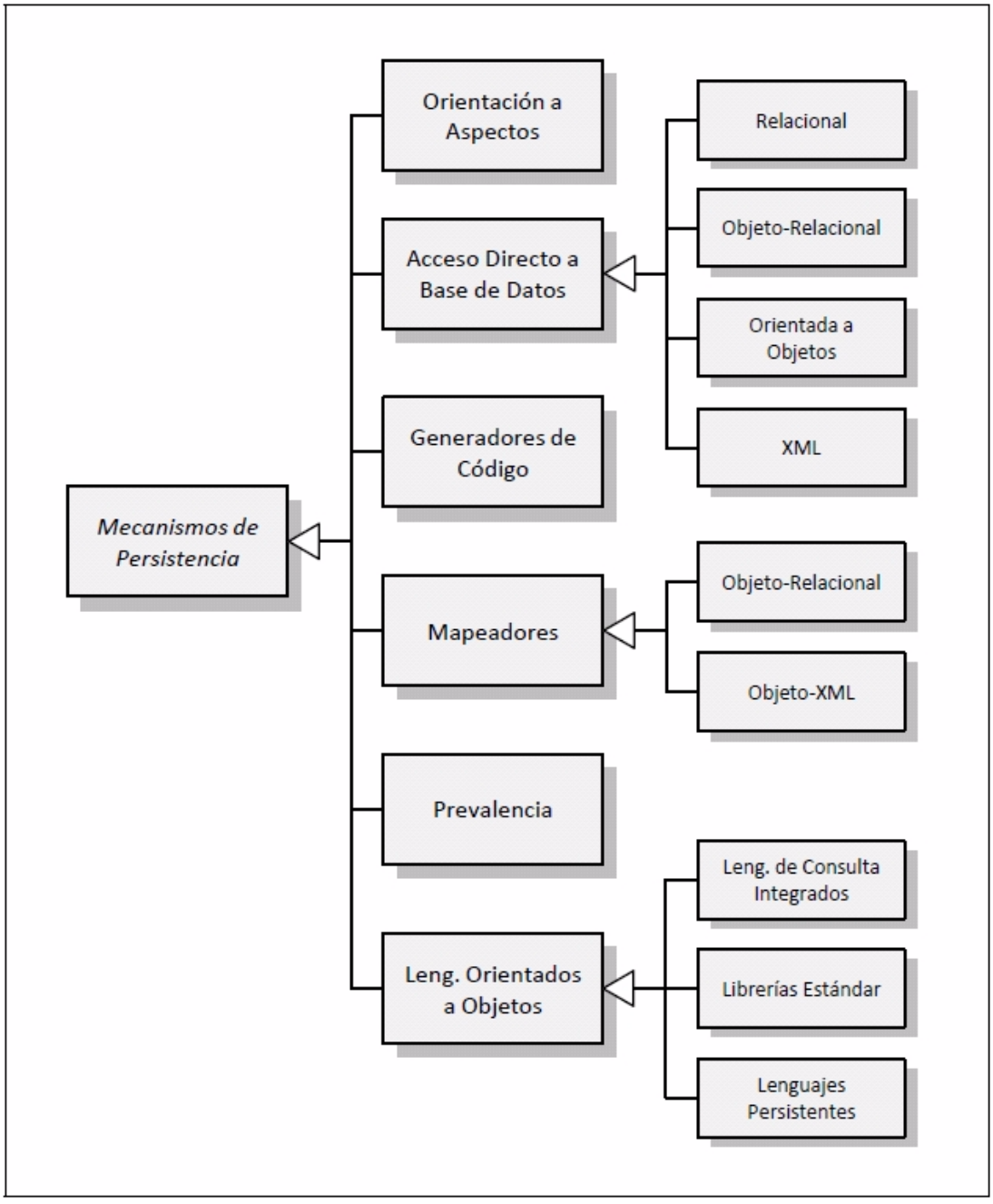


Figura 8: Mecanismos de persistencia, agrupados por herencia

técnica base (p. ej. en el caso de los mapeadores) tienen características diferenciadas (el caso de los mapeadores objeto-relacional y objeto-XML). Concretamente, la clasificaci´on propuesta define categorías que contemplan: Mapeadores (mapeo de objetos a otro modelo de datos), Acceso Directo a

Base de Datos (interacción directa con una base de datos), Lenguajes Orientados a Objetos (soporte del lenguaje para la persistencia de objetos), Orientación a Aspectos (implementación de persistencia a través de aspectos), Generadores de Código (generación del código para persistencia a través de herramientas) y Prevalencia (persistencia mediante snapshots y fuerte uso de memoria principal).

4.5.1 Acceso Directo a 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 DBMS. Téngase en cuenta que no existe ningún tipo de middleware entre la aplicación y el DBMS utilizado.

1. **Acceso Directo a Base de Datos Relacional.** Haciendo uso de 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.
2. **Acceso Directo a Base de Datos Objeto-Relacional.** Haciendo uso de 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.
3. **Acceso Directo a Base de Datos Orientada a Objetos.** Haciendo uso de una base de datos orientada a objetos como dispositivo 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.
4. **Acceso Directo a Base de Datos XML**. Haciendo uso de una base de datos XML como dispositivo de almacenamiento, es la intención fundamental proveer un mecanismo de persistencia que permita almacenar los datos de los objetos en documentos XML.

4.5.2 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. El conjunto de problemas que se derivan de estas diferencias recibe el nombre de impedance mismatch.

1. **Mapeadores Objeto-Relacional.** La intención 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. Debido a las diferencias entre la representación tabular de información y la encapsulación de los datos en objetos (problema conocido como impedance mismatch objeto-relacional) debe considerarse alguna estrategia (manual o automática) para poder resolver estas diferencias e integrar ambas tecnologías.
2. **Mapeadores Objeto-XML.** Es la intención de este mecanismo 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 (incluso sin la necesidad de un DBMS), al mismo tiempo de que se facilita el intercambio de dichos datos con otros sistemas.

A diferencia de los modelos como DOM y JDOM, los mapeadores objeto-XML considerados toman un enfoque centrado en los datos y no en la estructura del documento XML de forma tal que el desarrollador utiliza objetos de negocio que reflejan el contenido mismo de los documentos.

4.5.3 Generadores de Código

El surgimiento de numerosos frameworks que resuelven parte de la problemática del desarrollo de una aplicación empresarial y el creciente uso de patrones ha permitido el desarrollo de aplicaciones empresariales más robustas en menos tiempo. Sin embargo el programador aun se ve obligado a realizar tareas repetitivas. La idea de que el programador debe concentrarse en el código que representa la lógica de negocio es la principal motivación para el surgimiento de los generadores de código. Concretamente, se trata de herramientas que basándose en metadata de un proyecto son capaces de generar código correcto, robusto y que aplica los patrones de diseño pertinentes. Este código generado se encarga de resolver parte de la problemática del sistema, en este caso la persistencia del mismo.

4.5.4 Orientación a Aspectos

La Programación Orientada a Aspectos (AOP) es un paradigma que permite definir abstracciones que encapsulen características que involucren a un grupo de componentes funcionales, es decir, que corten transversalmente al sistema (p.ej. seguridad, comunicación, replicación, etc.). 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.

4.5.5 Lenguajes Orientados a Objetos

La categoría de lenguajes orientados a objetos como mecanismo de persistencia implica resolver la persistencia de datos utilizando funcionalidades provistas por el propio lenguaje de programación.

1. **Librerías Estándar.** La idea fundamental de este mecanismo yace en utilizar las funcionalidades básicas incluidas en la infraestructura de un lenguaje de programación orientado a objetos a fin de resolver la persistencia de un sistema. Se trata luego de una estrategia simple que evita la inclusión de frameworks u otras herramientas ajenas al lenguaje. Tomando como ejemplo el lenguaje de programación Java, se tienen librerías para el manejo de archivos que permiten persistir los datos de una aplicación en forma de texto plano, CSV (Comma Separated Value) o XML. Asimismo se pone a disposición del desarrollador la posibilidad de utilizar técnicas de serialización de objetos.
2. **Lenguajes Persistentes.** El objeto de este mecanismo implica resolver de manera transparente la persistencia de datos haciendo uso de funcionalidades provistas por el lenguaje de programación, cumpliendo con los tres principios de persistencia ortogonal (o en su defecto, una relajación de los mismos): ortogonalidad de tipo, persistencia por alcance e independencia de persistencia. En concreto, dichos principios implican respectivamente que: (1) Cada objeto, independiente de su tipo, tiene el mismo derecho a ser persistido. (2) Se persisten solo aquellos objetos que son alcanzables desde la raíz de persistencia. (3) La introducción de persistencia no puede introducir cambios semánticos en el código fuente.
3. **Lenguaje de Consulta Integrado.** Existen múltiples interfaces de acceso a datos que son utilizadas desde numerosos lenguajes de programación como método para administrar y consultar los datos persistentes de un sistema. El uso de interfaces de este tipo tiene como consecuencia una disminución en la claridad de la solución. Esto debido no solo al agregado en el código fuente de llamadas a operaciones definidas por el API en particular, sino también por la inclusión, en forma ajena al lenguaje de programación, de lenguajes de consulta como

ser SQL, OQL, HQL, etc. Tomando en cuenta lo anterior, el mecanismo propone la utilización de un lenguaje de programación que integre como parte del mismo un lenguaje de consulta y administración de datos persistentes.

4.5.6 Prevalencia

La prevalencia de objetos surge como una propuesta diferente a utilizar bases de datos para lograr la persistencia. La mayoría de los programas operan sobre datos en memoria principal, ya que de esa forma logran procesar los datos más rápido que en un escenario donde se trabaja directamente en memoria secundaria. Este mecanismo permite mantener las instancias de objetos que componen a un sistema en memoria principal, añadiendo en forma periódica, la persistencia del estado de los anteriores mediante técnicas de serialización. La persistencia del estado de los objetos se denomina snapshot de los objetos en memoria, y representa el último estado consistente de los datos. Asimismo, permite el manejo de transacciones y la capacidad de recuperar un estado consiste del sistema tras la caída del mismo.

5 Frameworks de persistencia

La persistencia de datos en Java viene facilitada por mapeadores Objeto/Relacionales (O/R). Estas tecnologías conforman técnicas de programación para enlazar un lenguaje de programación orientado a objetos con una base de datos relacional. Los mecanismos de mapeo O/R permiten al programador mantener una perspectiva orientada a objetos y cuidar que se consiga aportar una solución a la lógica de negocio, eliminando el obstáculo que supone compatibilizar objetos con modelos relacionales. El framework para la persistencia de datos se ocupa de todos los detalles que conllevaría desarrollar un mecanismo de mapeo personalizado.

Hoy en día, uno de los asuntos más debatidos y discutidos más apasionadamente en la industria del software es: ¿cuál de las tecnologías de persistencia (o frameworks para el mapeo Objeto/Relacional) merece ser la alternativa dominante? Esta discusión ha creado una división en la comunidad Java. Mientras esto siga así, dicho debate seguirá en pie y los desarrolladores de software deberán elegir la tecnología que mejor se adapte a sus necesidades, y esto se debe extender a cualquier componente que conforme la arquitectura del sistema completo, incluyendo el driver JDBC.

Consecuentemente, la capa de mayor criticidad y por consiguiente en la cual más trabajo se ha desarrollado en los últimos años es la de persistencia. Debido al choque de impedancia que se produce entre los objetos del modelo de negocio y los datos persistidos en una base de datos relacional, es que esta capa requiere un tratamiento particular.

Gran parte de los productos que se han generado atacan el problema del mapeo y acceso a los datos persistentes. Algunos de los más conocidos son:

- EJB Entity beans
- JDBC
- SQLJ
- TopLink
- CocoBase
- Hibernate / nHibernate
- JPOX (JDO)
- Versant (JDO)
- OBJ
- Object Spaces

Debido a algunas limitaciones de EJB Entity Beans han surgido las otras alternativas. Básicamente los Entity Beans presentan la característica de ser usables cuando los modelos de dominio son simples, deben ser distribuidos y conectarse a otros sistemas. Su utilización es buena cuando existe un mapeo uno a uno con las tablas de la base de datos. Es decir cuando la granularidad es baja. No admiten herencia entre clases componentes. Por esta razón y debido a esta limitación han surgido otros productos. Debido a esto también es que se desaconseja su utilización.

Por ejemplo, Toplink es un producto muy utilizado en el mercado, ahora integrado al servidor de aplicaciones Oracle 9i AS. Permite entre otras cosas:

- Integración de EJB CMP
- Mapeo de objetos a múltiples tablas
- Uso de herencia
- Soporta bloqueo optimista de tablas
- Transacciones anidadas

- Permite adaptar el mapeo de datos entre objetos y tablas
- Permite realizar el diseño en ambas direcciones, objetos / tablas y tablas / objetos.
- Permite adaptar código SQL generado
- Permite el uso de Store Procedures
- Administra pool de objetos
- Tiene una desventaja, es dependiente de APIs propietarias.

Hibernate también es muy utilizado tanto en el ambiente java como .net.

JDO es una especificación java que se espera se convierta en el estándar de administración de bases de datos orientadas a objetos. No exige la implementación de interfaces. Presenta menor sobrecaga en la creación de objetos y su configuración se realiza en un archivo XML, siendo más simple que la de los EJB CMP.

Presenta ventajas que heredó de EJB tales como el encapsulamiento de datos de la aplicación, el mapeo a tablas, trabaja en transacciones delimitadas por los Session Beans, administra pool de objetos. También posee ventajas propias como trabajar con clases java genéricas y las hace persistentes, requiere menor infraestructura.

5.1 Frameworks de persistencia de código libre

Hibernate

Hibernate busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos: El usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Hibernate ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente.

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente Hibernate Annotations que implementa el estándar JPA, que es parte de esta plataforma.

- Herramienta libre bajo licencia LGPL
- Es una herramienta madura, creada en el 2001, y es una de las más extendidas.
- Es un motor de persistencia para Java, aunque hay una version para la plataforma .net llamada Nhibernate.
- Compuesto de muchos proyectos como: core (proyecto principal), annotations, entity manager, shards, validator ...
- Tiene un montón de plugins para eclipse y tareas de Ant para ayudar a la utilización y automatización de la persistencia. Están dentro del proyecto hibernatetools
- Hibernate soporta paradigmas de la orientación a objetos como son el polimorfismo y las asociaciones.
- Ofrece un lenguaje de consultas de datos llamado HQL (Hibernate Query Language).
- Se pueden crear filtros definidos por el usuario.
- Gracias al proyecto HibernateAnnotations se pueden utilizar anotaciones de JDK 5.0 junto con los ficheros xml para el mapeo de objetos.

OJB

ObJectRelationalBridge (OJB) es una herramienta de correspondencia de objeto / Relational que permite mantener una transparencia en la persistencia contra SGBS para objetos Java. OJB ha sido diseñado para uno conjunto grande de aplicaciones, de sistemas empotrados a aplicaciones 'rich client', hasta arquietecturas J2EE multinivel. OJB se integra con facilidad en servidores de aplicación J2EE. Soporta la consulta de JNDI datasources. Se pone a la venta con JTA completo y la integración de JCA. OJB puede ser usado dentro de JSPs, Servlets y SessionBeans. OJB provee el soporte especial para Bean Managed EntityBeans (BMP).

Torque

Torque es una capa de persistencia, las herramientas que incorpora permiten crear un modelo físico de datos adaptado a diferentes SGBD gracias al empleo de XML.

Además permite evitar la codificación directa de sqls, así como abstraer al programador de la problemática vinculada con la apertura y cierre de conexiones a BBDD.

Es una herramienta muy potente que se debe usar con sumo cuidado, puesto que si se usa de manera inadecuada puede evitar los inconvenientes del acceso a BBDD a costa de introducir nuevos

problemas.

El Torque incluye un generador para generar todos los recursos de base de datos requeridos por su aplicación e incluye un ambiente de tiempo de ejecución para dirigir las clases generadas. El IDE en tiempo de ejecución del Torque incluye todo que usted necesita para usar las clases de OM / Peer generadas. Incluye un pool de conexión con jdbc.

Castor

Castor es un framework de datos Open Source para Java. Es la ruta más breve entre objetos de Java, documentos de XML y tablas de BBDD relacionales. Castor provee un vinculo de gestión de persistencia entre Java hacia XML, Java hacia SQL, y otros.

Cayenne: Professional Object Relational Mapping.

Cayenne es un framework poderoso y completo para correspondencia entre Objetos Java y Mapeo Relacional. Es open Source y totalmente libre. Una de las diferencias de Cayenne principales son que viene con herramientas de modelado GUI, para plataformas cruzadas. Esta característica coloca a Cayenne en su propia liga, haciéndolo una elección muy atractiva sobre tantos productos de comerciales de código cerrado.

- Herramienta de código abierto de la fundación Apache.
- Portabilidad entre las bases de datos JDBC.
- Funcionalidad de cache, para hacer el acceso a la base de datos más eficiente
- Capacidad de paginación para cargar de la base de datos solo los objetos que se necesitan en el momento.
- Tutorial web paso por paso para aprender el manejo de esta herramienta.

TriActive JDO (TJDO)

- Implementación libre de la especificación JDO 1.0.1 de Sun.
- Implementa el lenguaje de consultas JDOQL, aunque permite consultas SQL directas.
- Auto-creación de todos los elementos del esquema necesarios (tablas, claves , índices).
- Auto-validación de la estructura del esquema en tiempo de ejecución.
- Capacidad de mapear clases de Java a Vistas SQL.

Jaxor

Jaxor es una herramienta de generación de código o de mapeo que toma la información definida en XML relacionando las entidades del modelo relacional para mapearlas y generar clases, interfaces y objetos y buscar objetos que puedan ser usados en cualquier aplicación Java (incluyendo JFC / Swing, J2EE y herramientas en linea de comando). Su gran velocidad de generación de código se debe al uso estructurado de plantillas y la generación mediante mecanismos de conversión fijos.

JDBM

JDBM es un motor de persistencia transactional para Java. Aspira a ser para Java lo que GDBM es para otros lenguajes (C / C + +, Python, el lenguaje perl, etc.): Un motor de persistencia rápido y simple. Todas actualizaciones son hechas en una manera transaccional segura. JDBM también provee estructuras de datos escalables, como HTree y B+Tree, para soportar la persistencia de grandes colecciones de objetos.

pBeans

PBeans es una capa de mapeo de BBDD Objeto/Relacional (O/R). Está diseñado para ser simple de usar y automatizado totalmente. La idea es que se ahorre tiempo y esfuerzo concentrándose solo en el desarrollo de clases Java, y no preocupándose por el mantenimiento o correlación de los scripts SQL, esquemas XML basados en, ni siquiera en la generación de código. El framework pBeans se cuida de tomar muy poca ayuda del desarrollador para concretar la persistencia de los JavaBeans.

Simple ORM

SimpleORM es un proyecto Open Source para mapeo Objeto / Relacional en Java (la licencia de estilo de Apache). Provee una implementación simple pero eficaz de mapeo O/R sobre JDBC, de bajo costo. No necesita de configuración mediante archivo XML.

Más que Java puro, SimpleORM es Java 100% depurado. Sin pre-procesamiento ni post-procesamiento de bytecode. Este framework simple y elegante no requiere ningún truco ingenioso. SimpleORM tiene mucho cuidado con semántica de BBDD. Bloqueo, calidad de la información, y almacenamiento son siempre tratados apropiadamente.
Java Ultra-Lite Persistence (JULP)
Un framework de muy pequeño tamaño.
Prevayler
El concepto de prevalencia implica el mantenimiento de la totalidad de las instancias de objetos que componen un sistema en memoria principal. La persistencia se realiza mediante snapshots del estado del sistema que se efectúan de forma periódica y que utilizan la técnica de serialización de los objetos en memoria. La primera implementación de prevalencia se desarrolló como un proyecto Java de código abierto, denominada Prevayler. Prevayler que incluye funcionalidades como ser rollback automático ante fallos, volcados XML, y mejoras en la escalabilidad global.
JPOX Java Persistent Objects
JPOX es una puesta en práctica de los objetos de los datos de OpenSource Java (JDO) que proporciona la persistencia transparente de los objetos de Java. JPOX pone las especificaciones en ejecución JDO1 y JDO2 y pasa los boths TCKs. JPOX apoya datastores de RDBMS. JPOX está poniendo JPA en ejecución 1 (EJB 3). Con una puesta en funcionamiento versátil y de gran rendimiento, JPOX está sobre la vanguardia de las puestas en funcionamiento de objetos (JDO) de datos de Java disponibles, ofreciendo uno JDO dócil gratis, que se puso en funcionamiento liberada bajo una licencia Open Source. JPOX 1.0 es completamente dependiente de JDO 1.01, y JPOX 1.1 tiene muchas características previas de JDO 2.0.
Ibatis SQL Maps
Maps de SQL de iBATIS provee unos medios muy simples y flexibles de cambiar de lugar los datos entre sus objetos de Java y una BBDD relacional. El mapeo de SQL realizado por el framework ayuda a reducir la cantidad de código Java que normalmente hay que realizar para acceder a una BBDD relacional. Este framework traza un mapeo entre JavaBeans a sentencias de SQL usando un descriptor de XML muy simple. La sencillez es la ventaja más grande del mapeo SQL sobre los otros frameworks y otras herramientas de correspondencia O/R. Para usar SQL solo necesita estar familiarizado con JavaBeans, XML y SQL. Hay muy poco más para aprender. No hay ningún plan complicado requerido para unir tablas o ejecutar consultas complicadas. Usando mapeo SQL se tiene el pleno poder de SQL. El framework de Mapeo de SQL puede trazar un mapa de casi cualquier base de datos a cualquier modelo de objeto y es muy tolerante con los diseños anticuados, o incluso diseños malos. Esto se consigue sin tablas de base de datos especiales o generación de código.
Smyle
Smyle suministra un enfoque innovativo y pragmático para almacenamiento de datos fácil de comprender, fácil de uso, pero lo suficientemente fuerte para aplicaciones del mundo real. Smyle es Open Source y bajo licencia LGPL.
Speedo
Speedo es una Open source implementación de la especificación JDO.
XORM
XORM es una capa de correspondencia O/R extensible para aplicaciones Java. Suministra la interfaz para persistencia basada en SGBDs mientras permite que desarrolladores se concentren en el modelo de objeto, no la capa física.
JDBCPersistence
JDBCPersistence es framework de mapeo de persistencia O/R. Es diferente de sus semejantes en lo que respecta a genera el bytecode requerido para mapear una clase en una tabla. Ha sido creado con los siguientes requisitos en mente:

• Ser rápido para cargar • Soporte para CLOBs y BLOBs • Cargar objetos persistentes desde java.sql.ResultSet • Tenga API compacto • Tenga dependencias mínimas sobre otros proyectos • Configuración de soporte vía API

Tabla 3: Frameworks de implementación en código libre

5.2 Frameworks de persistencia propietarios

JDO Genie
Genie de JDO es una implementación de JDO de máximo rendimiento que soporta BBDD relacionales de comerciales y open source. JDO Genie soporta JBoss, WebLogic, WebSphere, Tomcat, otros motores servlet y aplicaciones de 2-niveles.
LiDO
A diferencia de otras herramientas de correspondencia O/R, LiDO no sólo provee el acceso a BBDD relacionales, sino a cualquier otra fuente de datos a las que las demás no pueden acceder, como XML, Mainframe, ODBMS y texto plano.
JDO Toolkit
JDO es un componente básico fundamental para cualquier aplicación. Es la manera usual de realizar la persistencia de objeto en Java, tal y como los servlets son la manera usual de hacer request/response.
FrontierSuite
FrontierSuite, uno de los mejores frameworks de persistencia para guardar objetos de Java en RDBMS. Especializado en el manejo de la persistencia para aplicaciones de empresa, proporcionando motores de persistencia, mapeo O/R y soluciones de almacenamiento ObjectFrontier es diferente en la singularidad de su producto. Y la especialidad está sostenida por su compatibilidad objeto/relacional, una estrategia con la que pocos han sido exitosos.
JRelay
Almacena objetos gráficos de Java en SGBD relacionales con JRelay. JRelay usa las tecnologías más avanzadas en correspondencia O/R para conseguir máximas prestaciones y gran rapidez en la persistencia de objetos (JDO). JRelay combina la experiencia en capas objeto / relacional con la universalidad de un estándar abierto, que asegura la escalabilidad, la robustez y el futuro soporte para uno de las partes más críticas en toda aplicación: el manejo de la persistencia de datos de una aplicación.
IntelliBO
• Soporte para servidores de aplicaciones: WeLogic, WebSphere, SAP WAS, Oracle, Jboss, Apache Geronimo, Tomcat ... • Soporte para bases de datos JDBC genéricas, y además: Oracle, DB2, MS SQL, informix, HSQLDB, postgreSQL, mySQL .... • Tiene muchas opciones avanzadas para el mapeo de objetos, como división de un objeto simple en múltiples tablas, estrategias de mapeo de herencia (TablaSubclase, Tablasuperclase, Tablanueva), mapeo serializado, mapeo agregado.. • Opción de verificar las clases y las instrucciones de mapeo. • Opción para comprobar la consistencia de la estructura de tabla actual. • Generador automático de código fuente de las clases e instrucciones de mapeo a partir de las estructuras de tablas. • Generador automático de las estructuras de tablas a partir de las clases e instrucciones de mapeo. • Integración con Borland Jbuilder, Apache Ant, Eclipse y IBM WebSphere Application Developer. • Ejemplos de proyectos y documentación comprensible. • Edición profesional (de pago), edición comunidad (gratuita), plugin para eclipse.
KodoJDO

• Motor de persistencia para Java basado en el código de OpenJPA de Apache (JPA= Java • Persistence Api). • Soporte para la especificación JDO 2.0 y EJB 3.0 JPA. • Herramienta comercial. • Es capaz tanto de crear un esquema nuevo a partir de las clases, las clases a partir del esquema, o mapear clases existentes en un esquema existente. • Diseñado para trabajar con o sin aplicaciones servidor. • Soporte para multiples bases de datos: Oracle, IBM DB2 y Informix, Microsoft SQL server ... • Integración con Eclipse, Borland Jbuilder, IBM WSAD y NetBeans. • Soporte para servidores de aplicaciones J2EE: Weblogic, WebSphere, JKBoss...
Oracle
• Motor de persistencia para POJO (Plain Old Java Objects) ejb 2.1, cmp and bmp • Implementa la api JPA enfocada a la estandarización del la persistencia objetorelacional. • Permite persistencia de objetos Java en bases de datos relacionales accesibles utilizando los drivers JDBC • También permite la persistencia de objetos java en bases de datos objetorelacionales como las bases de datos de Oracle. • Soporta Enterprise information system (EIS); permitiendo la persistencia de objetos java a fuentes de datos no relacionales que utilizan la arquitectura J2C (J2EE Connector architecture adapter). • Conversion entre objetos Java y documentos del esquema XML (XSD), usando la arquitectura java para XML (JAXB) • Soporte para servidores de aplicaciones IBM Websphere y BEA Weblogic. • Soporta el cache de objetos para mejorar el rendimiento.

Tabla 4: Frameworks de implementación propietarios

5.3 Comparación frameworks que implementan JDO

Características	FrontierSuite	Kodo	Lido	OpenFusion	TJDO	OJB
Especificación	1,0	2,0	1,0	1,0	1.0.1	1,0
RDBMS	Si	Si	Si	Si	Si	Si
ODBMS	No	Si	Si	No	No	No
Opciones especificación	Si	Si	Algunas	Algunas	Algunas	No
Entorno Administrativo	Si	Si	Si	Si	No	Si
Adaptable a entornos de programación	Si	Si	Si	No	No	No
Correspondencia reversible	Si	Si	No	No	No	Si
Versión evaluación	Completa	Completa	no es mantenida	No	Si	Si
Opciones propias	Si	Si	Si	Si	No	Si
MultiBase	Si	Si	Depende del fabricante	Si	Si	Si

Tabla 5: Implementación de JDO, comparativa

Cuadro Comparativo de Mapeadores

Criterio	Hibernate	JPA/TopLink	LLBLGen Pro
	Nivel de Productos		
Ultima Versión	v3.2.1	v2.0b41	v2.0
Fecha de Lanzamiento	16/11/06	11/05/06	02/07/06
Licencia	LGPL	CDDL	Comercial
Plataformas	Múltiples (Java)	Múltiples (Java)	Múltiples (.NET)
	Nivel de Mapeadores		
Arquitectura de *Cache*	2 Niveles	2 Niveles	2 Niveles
Bulk Data Manipulation	✓	✓	✓
Consultas Nombradas	✓	✓	✓
Claves Primarias Compuestas	✓	✓	✓
Construcción Manual de SQL	✗	✗	✗
Construcción `GROUP BY`	✓	✓	✗
Constructor por Defecto	✓	✓	✗
Consultas Dinámicas	✓	✓	✓
Consultas Polimórficas	✓	✓	✓
Creación Automática de Esquema	✓	✓	✗
DBMSs Soportados	JDBC-compliant	JDBC-compliant	SQL Server, Oracle, PostgreSQL, Firebird, DB2 UDB, MySQL, MS Access
Estrategias de *Fetching*	{eager, lazy}	{eager, lazy[1]}	{eager, lazy}
Estrategias para Herencia	{table-per-class, table-per-hierarchy, table-per-subclass}	{table-per-class, table-per-hierarchy, table-per-subclass}	{table-per-class, table-per-hierarchy, table-per-subclass}
Funciones Agregadas	{avg, count, max, min, sum}	{avg, count, max, min, sum}	{avg, count, max, min, sum}
Generación Automática de ID	✓	✓	✓
Lenguaje de Consulta OO	✓(HQL)	✓(JPQL)	✗
Mapeo de 1 Clase a N Tablas	✓	✓	✗
Mapeo de N Clases a 1 Tabla	✓[2]	✓[2]	✓
Objetos Asociados con *Outer Joins*	✓	✓	✓
Ortogonalidad de Tipo	✓	✓	✓
Persistencia de EJBs	✓	✓	✗
Persistencia con Campos Privados	✓	✓	✓
Persistencia con `get/set`	✓	✓	✗
Proceso/Generación de Código	✗	✗	✓
Relaciones Soportadas	{1-1, 1-n, m-n}	{1-1, 1-n, m-n}	{1-1, 1-n, m-n}
Soporte de *Annotations*	✓	✓	✗
Soporte para *Cascading*	✓	✓	✓
Soporte para *Detached Objects*	✓	✓	✓
Soporte para *Optimistic Locking*	✓	✓	✓
Soporte para Paginación	✓	✓	✓
Tablas Extra en la Base de Datos	✓	✓	✓
Tipos Personalizados	✓	✓	✓
Uniquing	✓	✓	✓

[1] Depende de si el proveedor de persistencia utilizado lo soporta.
[2] Solo una clase puede modificar los datos, el resto se restringe a lectura.

Tabla 6: Comparación de los Mapeadores de O/R principales

5.4 Evaluación Comparativa

5.4.1 JDO vs. EJB

Desde la aparición de JDO se ha especulado que esta tecnología podría sustituir a las EJB de entidad. Para entender esta afirmación se debe examinar que es exactamente un EJB de entidad. Como ya se ha explicado, los beans de entidad se dividen en dos categorías: persistencia manejada por contenedor (CMP) y persistencia manejada por bean (BMP). Los beans BMP contienen un código que puede almacenar el contenido del bean en un almacén de datos permanente. Los BMP tienden a ser independientes y no forman relaciones directas con otros beans BMP. No sería correcto decir que los beans BMP pueden ser sustituidos por JDO, puesto que un bean BMP hace uso directo de código JDBC. Esto viola uno de los principios de diseño de JDO ya que esta tecnología pretende abstraer al usuario de codificar con JDBC.

Los beans CMP permiten manejar la persistencia al contenedor. El contenedor es cualquier servidor que esté ejecutando el bean, y se encarga de manejar todo el almacenamiento actual. Los beans CMP también pueden formar las típicas relaciones [1 – n] ó [n a m] con otros beans CMP.

La función de un bean CMP es muy similar a la de JDO. Ambas tecnologías permiten persistir datos con una perspectiva orientada a objetos ya que siempre se persisten objetos y evitan tener que conocer los detalles de cómo los objetos están almacenados. JDO y CMP también son capaces de manejar relaciones entre objetos. Por tanto sí se puede hablar de que JDO puede sustituir a CMP.

Se debe tener en cuenta el crecimiento de las tecnologías. Los beans CMP todavía necesitan aumentar su aceptación en la comunidad de programadores. La mayor parte de las mejoras y crecimiento en general de las EJBs ha sido en el área de sesión. CMP y JDO padecen los mismos problemas a la hora de ser acogidos ya que ambas tecnologías abstraen demasiado al programador de lo que realmente sucede a nivel SQL. A la hora de realizar consultas complicadas el programador debe dedicar mucho tiempo intentando descubrir cómo generar dicha consulta equivalente a la sentencia SQL. En estos casos los programadores preferirían haber programado en SQL desde un primer momento.

5.4.2 JDO vs. JDBC/SQL

Sustituir a JDBC no es exactamente lo que busca JDO de la misma forma que podría hacerlo con CMP. JDO puede ser realmente una buena capa en el nivel superior a JDBC. En la mayoría de las instancias de JDO, se debe especificar una fuente de datos JDBC que establezca una referencia a la base de datos que JDO va a estar manejando. Por tanto, comparar a JDO con JDBC es algo que se debe hacer si se duda entre usar directamente JDBC o permitir que JDO lo use en lugar del programador.

Por una parte JDO libera al programador de la tarea de construir consultas SQL y de distribuir entre los atributos de un objeto Java los resultados obtenidos en un result set. Si se considera que la mayor parte de consultas JDBC se realizan con el fin de dar valores a los atributos de objetos Java, JDO debería ser una alternativa a tener en cuenta, puesto que en lugar de ejecutar una consulta y copiar los campos desde el result set de JDBC a objetos Java, JDO se puede hacer cargo de todo ello.

Las críticas a JDO vienen precisamente por las grandes cantidades de accesos innecesarios que realiza para llevar a cabo su tarea. JDO debe coger su consulta JDOQL y convertirla en su consulta SQL correspondiente. Entonces, esta consulta SQL es enviada a la base de datos, los resultados son recibidos y almacenados en sus respectivos objetos. Si hubiera un gran número de relaciones entre los objetos, es muy fácil que, como consecuencia, JDO haya accedido a muchos más datos de los necesarios. Es evidente que JDBC siempre va a ser más rápido que JDO ya que es más directo. Será elección del programador si la comodidad en la forma de trabajar que le ofrece JDO compensa su peor rendimiento.

Otro de los aspectos discutidos de JDO es la posibilidad de sustituir SQL por JDOQL. Cuando se usa JDBC se debe acudir a SQL para componer las consultas mientras que con JDO se usa JDOQL. JDOQL es un lenguaje de consultas basado en Java. Por una parte, JDOQL es mucho más sencillo de componer que SQL, especialmente cuando nos referimos a consultas sencillas. Sin embargo, no existen muchos programadores que dominen JDOQL.

De momento, este problema va a seguir existiendo ya que, como se ha comentado anteriormente, JDO no ha sido muy acogido en la industria del desarrollo de software. La mayoría de los programadores dominan SQL, y además son capaces de construir consultas SQL muy optimizadas a pesar de tener una gran complejidad.

Para muchos programadores, una herramienta que crea consultas automáticamente no es de gran utilidad, sobre todo si nos referimos a JDOQL, que solamente actúa en aplicaciones Java. Antes o después SQL será sustituido, pero para ello tendrá que llevarlo a cabo una tecnología más universal.

5.4.3 Hibernate vs. JDO

Hibernate se caracteriza por su completa transparencia para el programador. Al contrario que JDO, Hibernate se encarga de todo el proceso de persistencia. No hay que pasar por la tarea de ejecutar nada parecido al JDOEnhancer. Hibernate se encarga de hacer todo el proceso transparente, ya que basta con añadir sus librerías a la aplicación y rellenar su archivo de configuración para asignar la base de datos con la que se va a trabajar. Una vez dispuestos los ficheros de mapeo, se puede trabajar con la misma facilidad con la que se codifica con cualquier librería Java.

Otro punto muy a su favor es que Hibernate mantiene la posibilidad de realizar sus consultas en SQL. El HQL es el lenguaje para consulta específico de Hibernate, al igual que el JDOQL es el lenguaje a través del cual se realizan las consultas cuando se trabaja con JDO. Para usar JDO, el JDOQL es indispensable, ofreciendo una gran comodidad al programador a la hora de componer consultas sencillas. Sin embargo, cuando se trata de realizar sentencias más complejas, un programador que domine SQL con un nivel alto seguramente eche de menos la alternativa estándar. Hibernate ofrece ambas posibilidades.

6 Diseño de un Framework de persistencia

6.1 Introducción

Para el desarrollo del framework de persistencia se ha utilizado como base el libro "*The Design of a Robust Persistence Layer For Relational Databases*" [Scott W. Ambler], así como la evaluación tanto de los requisitos como del código (cuándo este ha estado disponible) de algunos de los principales framework de código open source en la actualidad.

Como todo debe tener una denominación, a este trabajo he decidido denominarlo FPUOC, es decir, Framework de Persistencia UOC.

Básicamente se ha perseguido desarrollar un framework sencillo que abstraiga las características de la BBDD a la que se conecta y permita el encapsulamiento de las sentencias SQL hacia/desde la SGBD partiendo de un modelo genérico de comportamiento.

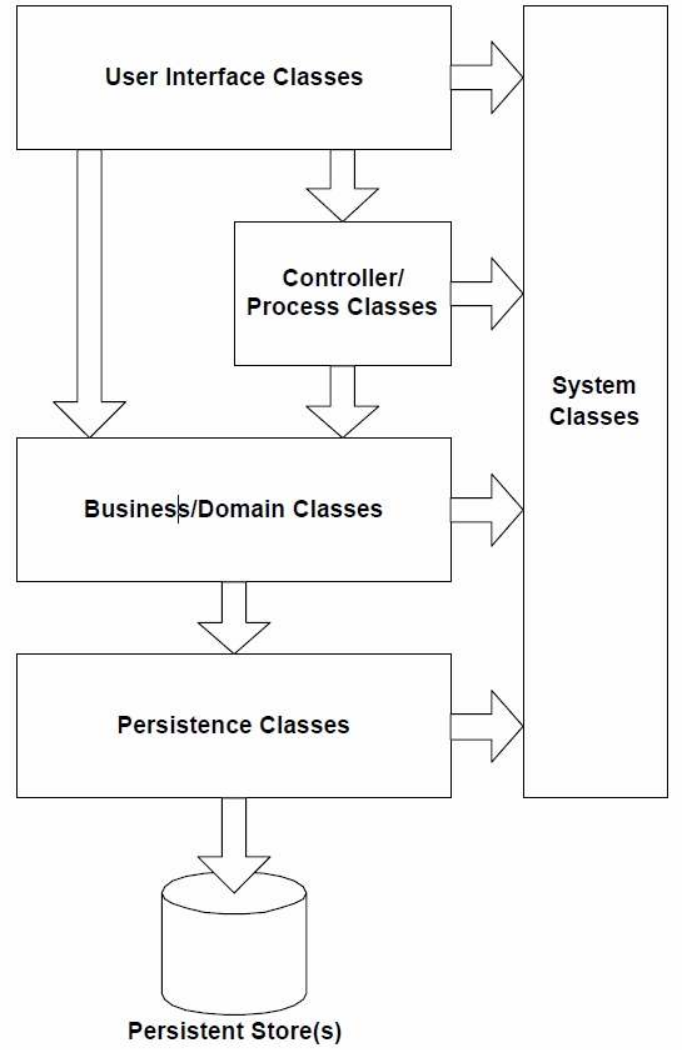


Figura 9: Arquitectura básica Framework persistencia [Scott W. Ambler]

6.2 Características de FPUOC

Las características que se han definido para este proyecto de desarrollo de framework de persistencia han sido las más sencillas y básicas, contando entre ellas con:

- Se deberá de poder ejecutar tanto en modo contenedor como de forma autónoma.
- El código deberá ser contenido, es decir, no deberá ser de gran tamaño, tipo Hibernate.
- Debe ser independiente de otros códigos, librerías o plataformas, excepto que será dependiente de JDK, es decir, que necesita del driver JDBC para poder funcionar.
- También será independiente de la plataforma SGBD, es decir, que no se desarrolla para ser ejecutado en una plataforma dependiente (como MySQL, ORACLE, SyBase,…).
- El requerimiento mínimo será de uso de JDBC 1.X o superior.
- Soportará la herencia directamente.
- Deberá poder soportar la implementación de varias clases por cada tabla de la BBDD.
- También deberá poder utilizar varias tablas del SGBD por cada clase implementada.
- El mapeo entre la BBDD y el framework deberá ser muy simple:
  <catalogo>.<esquema>.tabla.columna=nombreCampo

  Por ejemplo, el mapeo sería de la siguiente forma:

```
CREATE TABLE PRODUCTO(ID INTEGER NOT NULL PRIMARY KEY,NOMBRE VARCHAR,PRECIO DECIMAL)
...
public class Producto{
  Integer productoId;
  String nombreProducto;
  double precio;
...
}

...
ProductoFactory factory = new ProductoFactory(Producto.class);
Map mapping = new HashMap();
mapping.put("PRODUCTO.ID", "productoId");
mapping.put("PRODUCTO.NOMBRE", "nombreProducto");
mapping.put("PRODUCTO.PRECIO", "precio");
factory.setMapping(mapping);
...
```

Tabla 7: Esquema de mapeo de FPUOC

- Habilidad para utilizar persistencia mediante POJO.
- Uso del patrón DAO como elemento estructural del diseño.

6.3 Presentación de las clases FPUOC

El paquete org.fpuoc contiene 17 clases únicamente, de las cuales 4 son interfaces.

El modelo general de interacción, la arquitectura de comportamiento de las clases es el que se muestra en la figura 10:

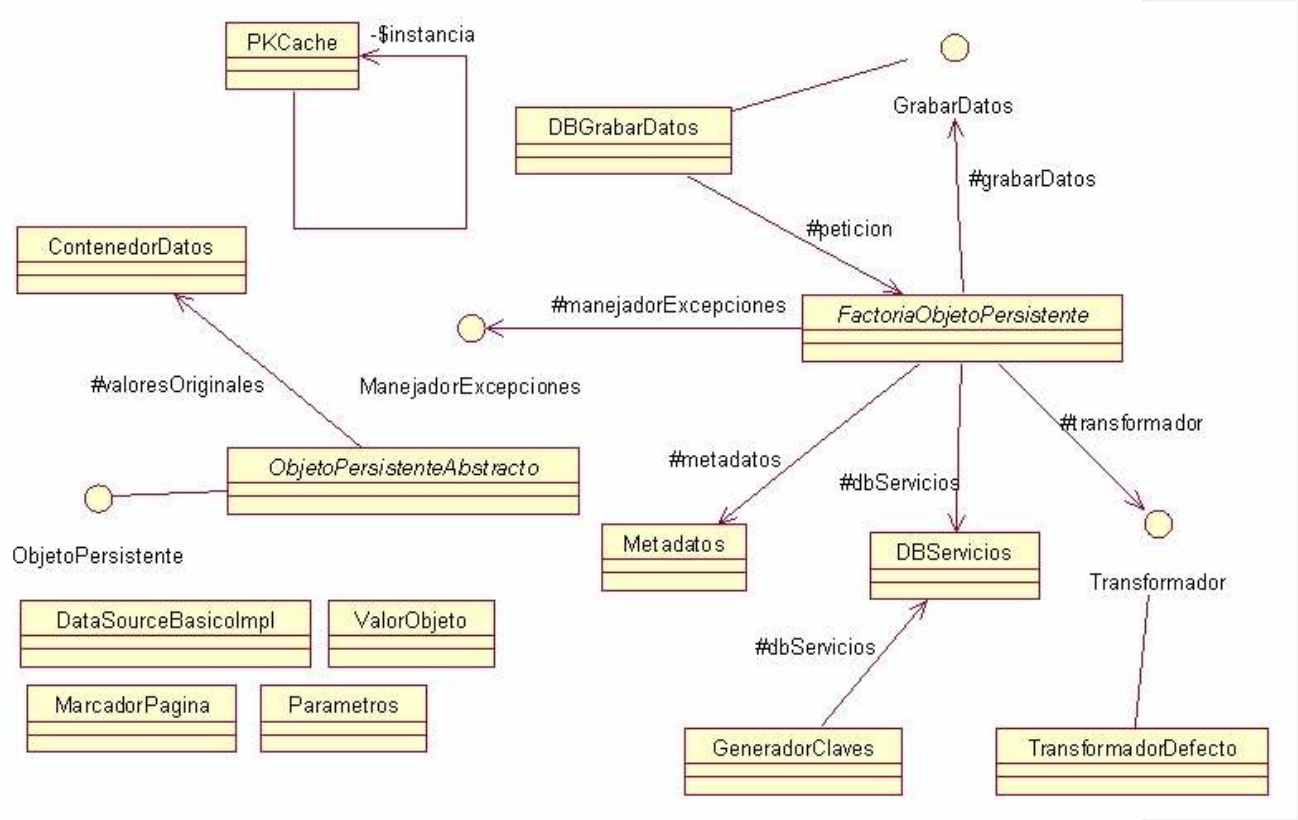


Figura 10: Arquitectura general del paquete org.fpuoc

6.4 Diseño de las clases de FPUOC

6.4.1 Diagramas de Jerarquía

Figura 11: Jerarquía de herencia de DBServicios

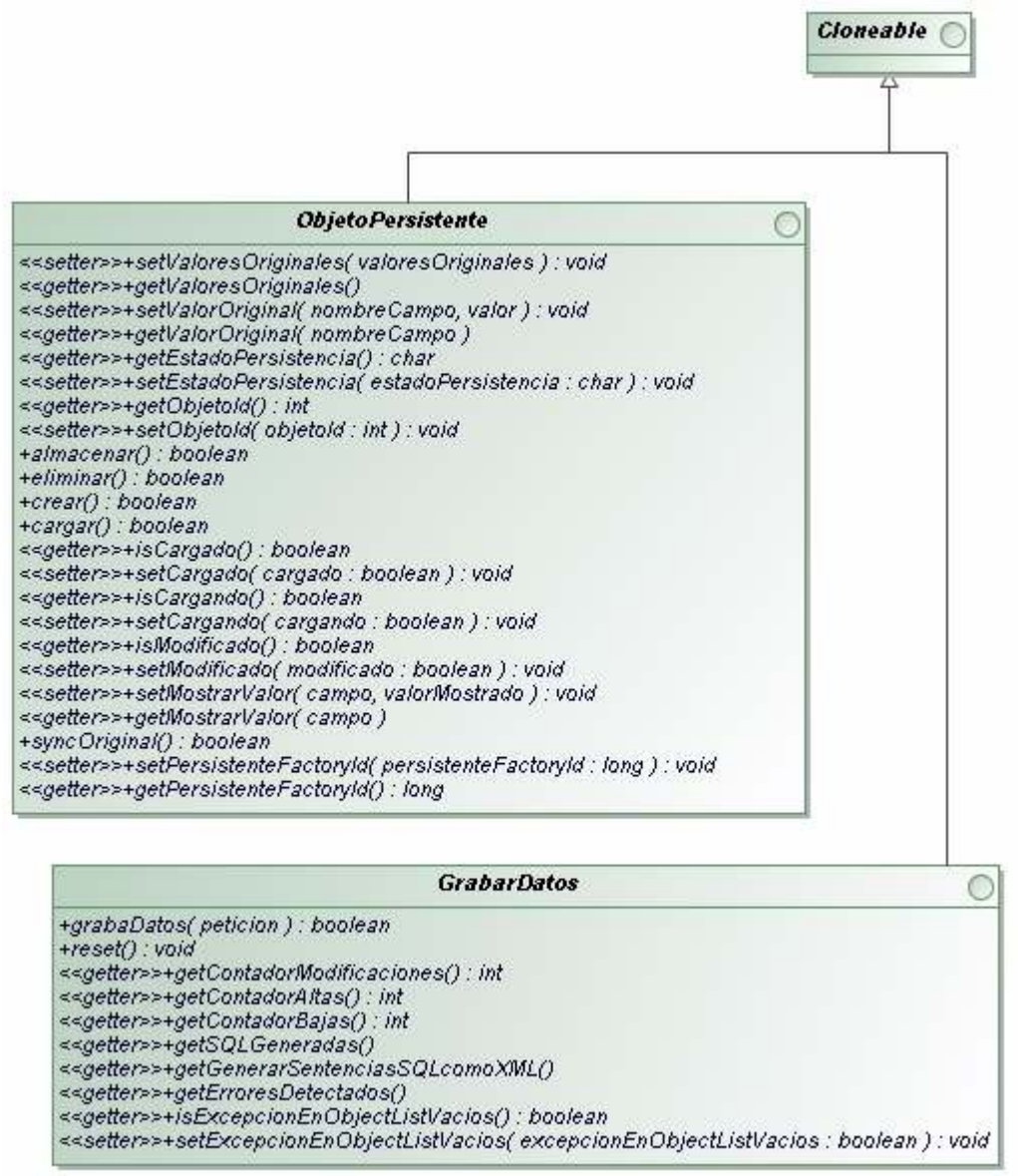


Figura 12: Jerarquía de herencia de ObjetoPersistente y GrabarDatos

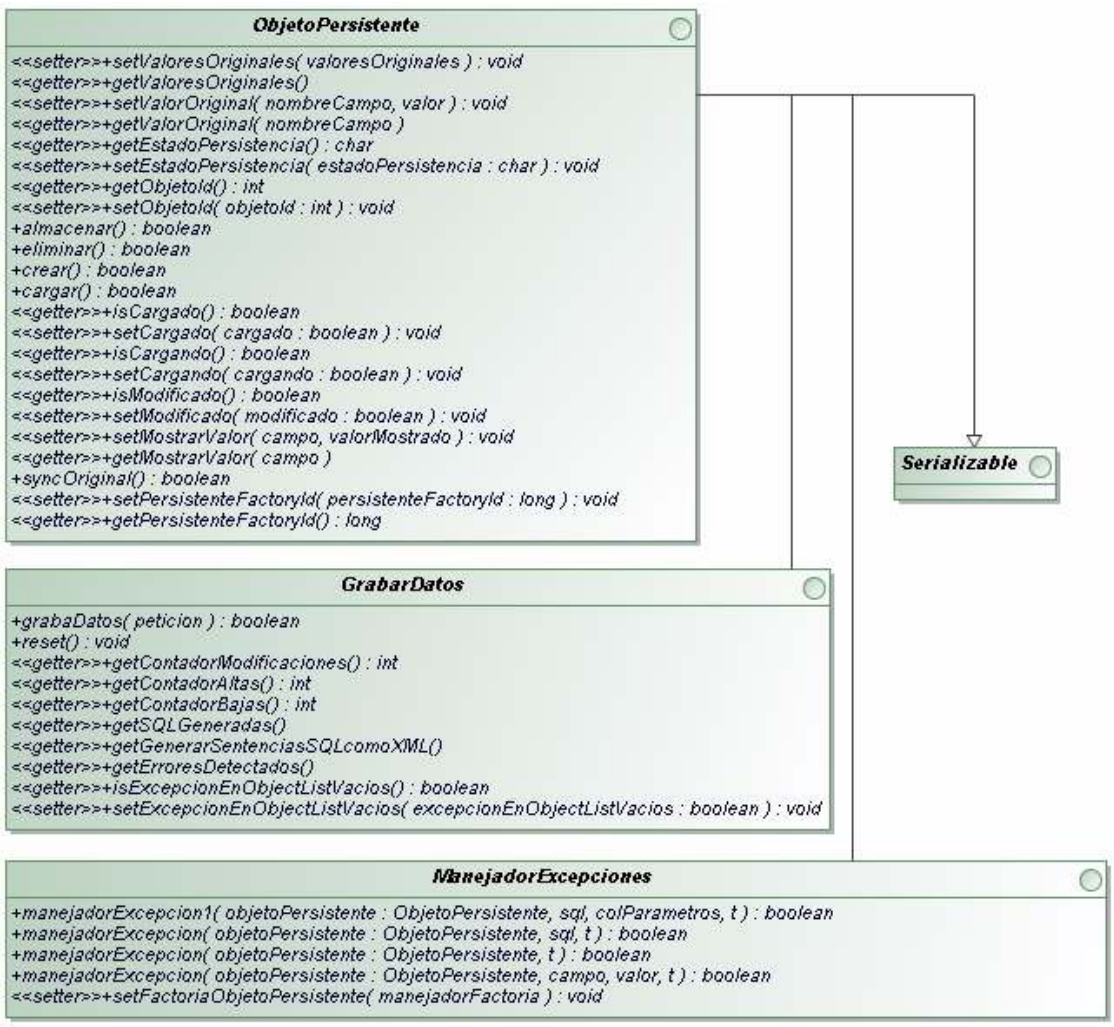


Figura 13: Jerarquía de herencia dependiente de Serializable

6.4.2 Diagramas de paquetes

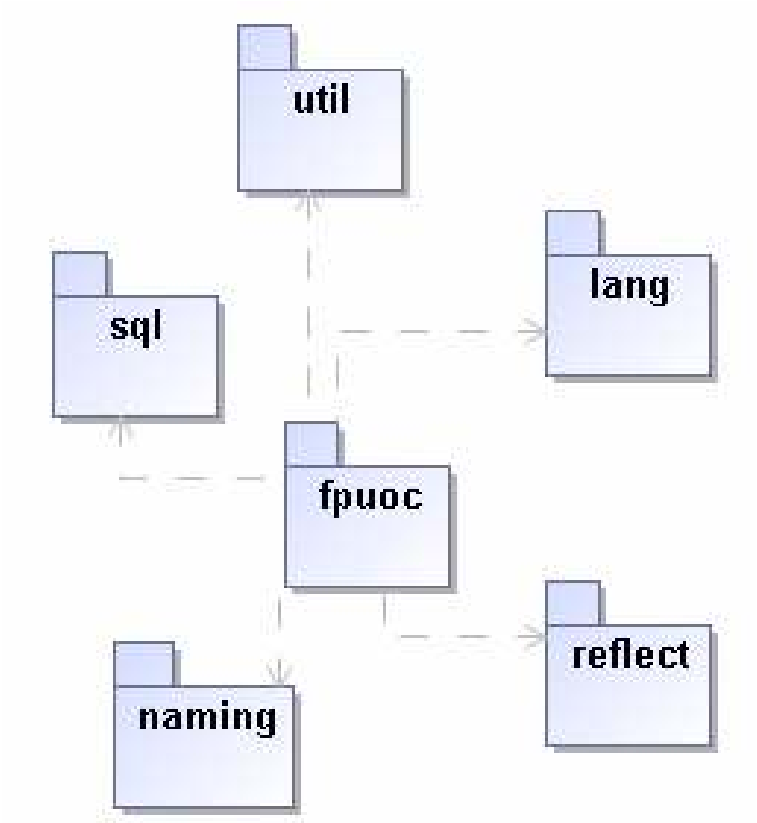


Figura 14: Estructura de paquetes básica de FPUOC

6.4.3 Diagramas de clases

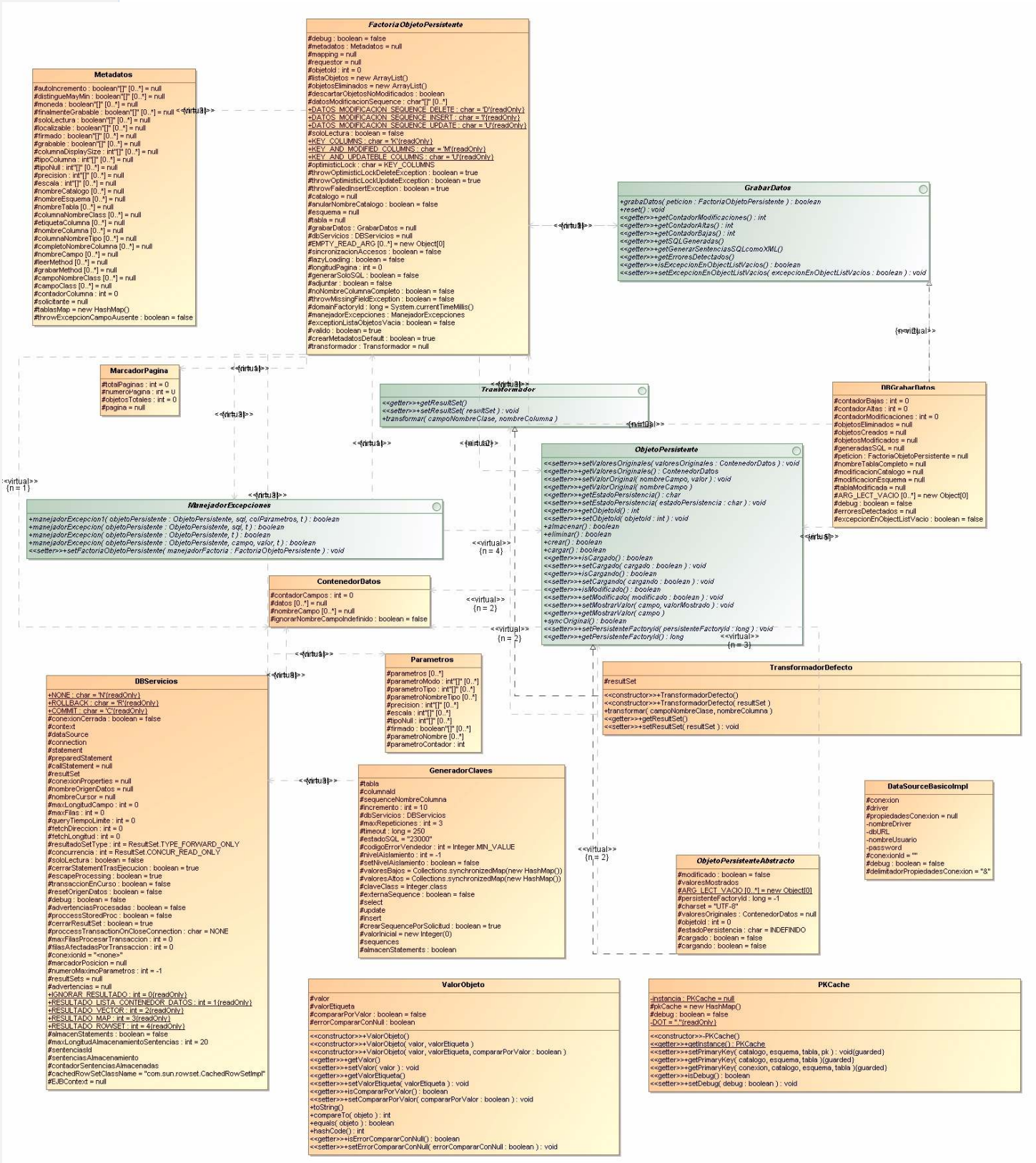
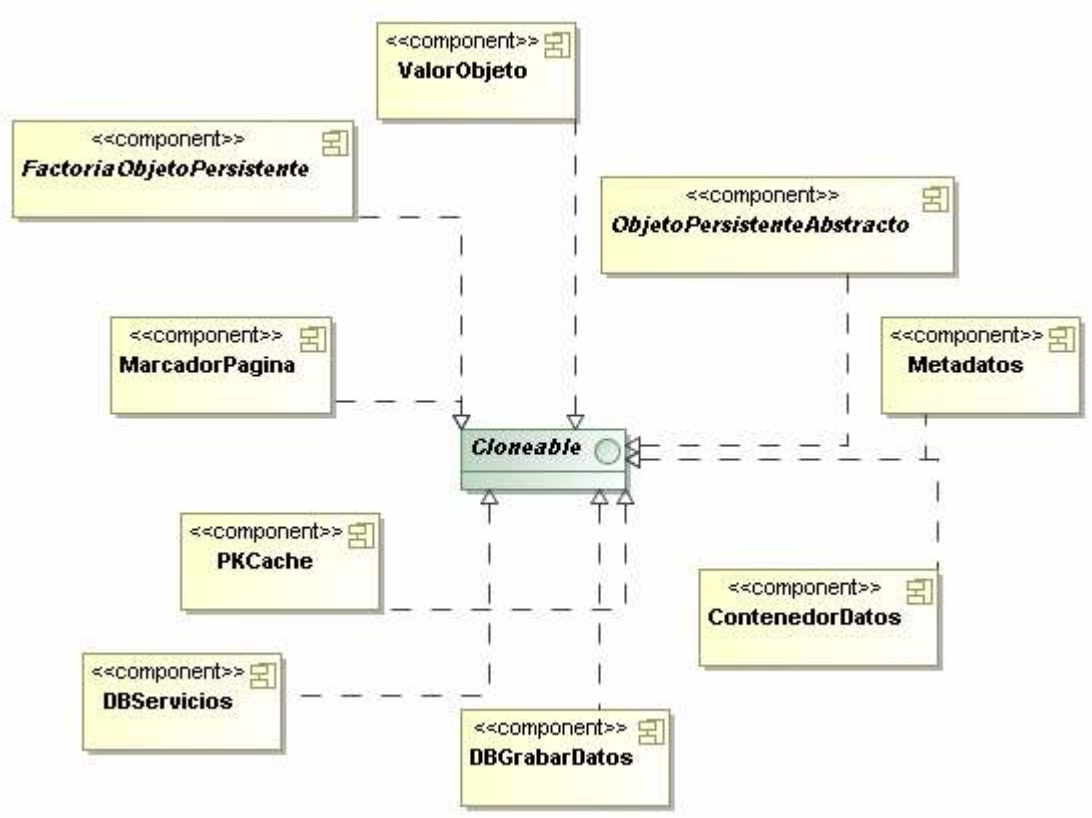
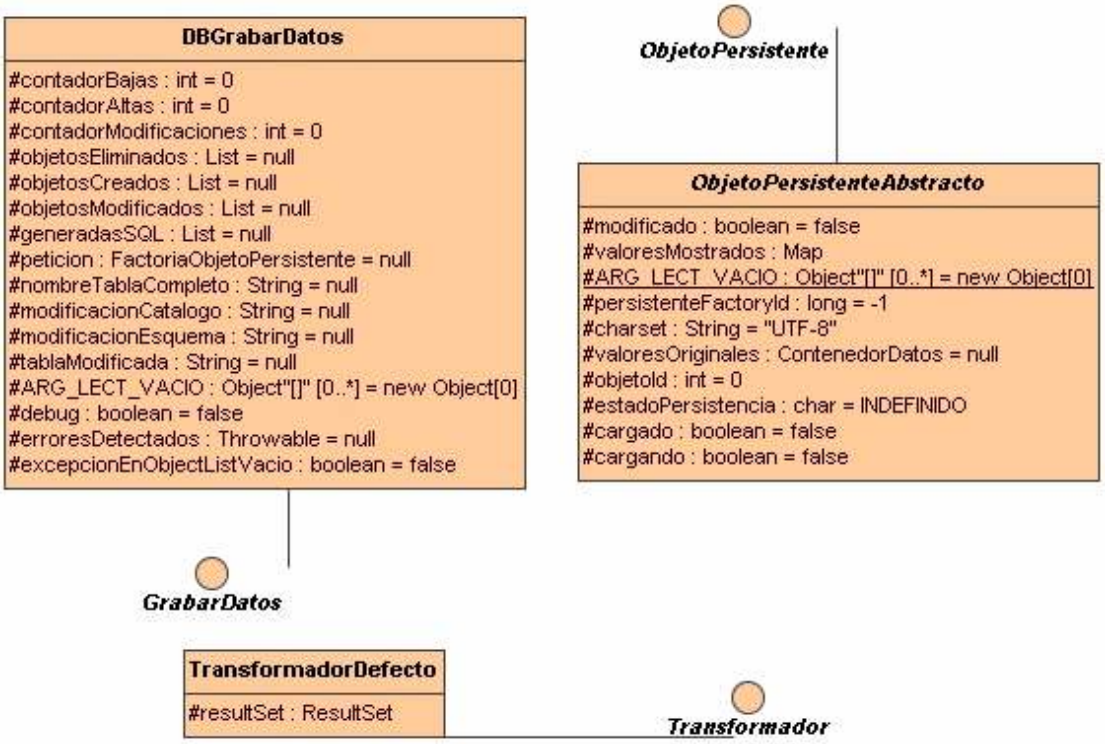


Figura 15: Diagrama general de dependencia de clases

6.4.4 Diagramas de componentes



6.4.5 Diagrama de realización de algunas clases



6.5 Uso de FPUOC

Para usar FPUOC se deben realizar las siguientes operaciones:

- El objeto que vayamos a implementar debe de extenderse (mecanismo de herencia necesario) de la clase org.fpuoc.ObjetoPersistenteAbstracto.
- A continuación se debe crear un objeto que extienda (herencia) la clase org.fpuoc.FactoriaObjetoPersistente.
- El paquete adicional de utilidades (org.fpuoc.utilidades) proporciona un método cómodo para realizar los dos pasos previos y además realizar un mapeo de propiedades sobre la BBDD.

El ejemplo a continuación está extraído del paquete org.fpuoc.ejemplos, que es el creado para explicar el uso del framework:

```
package org.fpuoc.ejemplos;

public class Cliente extends org.fpuoc.ObjetoPersistenteAbstracto implements java.io.Serializable, Cloneable{

  public Cliente() {}

  protected java.lang.Integer clienteId;
  protected java.lang.String nombre;
  protected java.lang.String apellidos;
  protected java.lang.String direccion;
  protected java.lang.String ciudad;

    public java.lang.Integer getClienteId() {
    return this.clienteId;
  }

  public void setClienteId(java.lang.Integer clienteId) {
    if (!isCargando()){
      if (clienteId == null){
        throw new IllegalArgumentException("Falta campo: clienteId");
      }
      if (!clienteId.equals(this.clienteId)){
        this.modificado = true;
      }
    }
    this.clienteId = clienteId;
  }

  public java.lang.String getNombre() {
    return this.nombre;
  }

  public void setNombre(java.lang.String nombre) {
    if (!isCargando()){
      if (nombre == null){
        throw new IllegalArgumentException("Falta campo: nombre");
      }
      if (!nombre.equals(this.nombre)){
        this.modificado = true;
      }
    }
    this.nombre = nombre;
  }

  public java.lang.String getApellidos() {
    return this.apellidos;
```

```
    }

    public void setApellidos(java.lang.String apellidos) {
        if (!isCargando()){
            if (apellidos == null){
                throw new IllegalArgumentException("Falta campo: apellidos");
            }
            if (!apellidos.equals(this.apellidos)){
                this.modificado = true;
            }
        }
        this.apellidos = apellidos;
    }

    public java.lang.String getDireccion() {
        return this.direccion;
    }

    public void setDireccion(java.lang.String direccion) {
        if (!isCargando()){
            if (direccion == null){
                throw new IllegalArgumentException("Falta campo: direccion");
            }
            if (!direccion.equals(this.direccion)){
                this.modificado = true;
            }
        }
        this.direccion = direccion;
    }

    public java.lang.String getCiudad() {
        return this.ciudad;
    }

    public void setCiudad(java.lang.String ciudad) {
        if (!isCargando()){
            if (ciudad == null){
                throw new IllegalArgumentException("Falta campo: ciudad");
            }
            if (!ciudad.equals(this.ciudad)){
                this.modificado = true;
            }
        }
        this.ciudad = ciudad;
    }
}
```

Tabla 8: Ejemplo herencia en el empleo de FPUOC: ObjetoPersistenteAbstracto

```
public class ClienteFactory extends org.fpuoc.FactoriaObjetoPersistente implements java.io.Serializable, Cloneable{

    protected Properties sqlMap = null;

    public ClienteFactory() {
        sqlMap = cargaMappings("Cliente.sql");
        initFactory();
        setRequestor(Cliente.class);
    }

     protected void initFactory(){
        try{
            DataSourceBasicoImpl ds = new DataSourceBasicoImpl();
            ds.setNombreDriver("org.hsqldb.jdbcDriver");
            ds.setDbURL("jdbc:hsqldb:hsql://localhost");
            ds.setNombreUsuario("sa");
            ds.setPassword("");
            this.dbServicios = new DBServicios();
            dbServicios.setDataSource(ds);
            setMapping(cargaMappings("Cliente.properties"));
        }catch (Exception e){
            e.printStackTrace();
```

```
    }
  }

  public Properties cargaMappings(String path){
    java.io.InputStream inStream = null;
    Properties props = new Properties();
    try{
      inStream = this.getClass().getResourceAsStream(path);
      props.load(inStream);
    }catch(java.io.IOException ioe){
      throw new RuntimeException(ioe);
    }finally{
      try{
        inStream.close();
      }catch(java.io.IOException ioe){
        throw new RuntimeException(ioe);
      }
    }
    return props;
  }

  public void select(){
    System.out.println("\n======================== Ejemplo de uso de varios metodos para rellenar collections
==========================\n");
    try{
      setMapping(cargaMappings("Cliente.properties"));
      String sql = sqlMap.getProperty("buscaTodosClientesConTelefono");
      System.out.println("============== como lista ValoresObjeto (primera columna valor, segunda contenido)
=================");
      List list = dbServicios.getResultadoComoValoresListaObjetos(sql);
      System.out.println(list);
      System.out.println("============== como Vector de Vectores ==================");
      Vector vector1 = dbServicios.getResultadoComoVector(sql);
      System.out.println(vector1);
      System.out.println("============== como Lista de tipo Map  (key=nombre columna, valor=lista valores por columna)
=================");
      Map map = dbServicios.getResultadoComoMap(sql);
      System.out.println(map);
      System.out.println("============== como array de ContenedorDatos ==================");

      ContenedorDatos[] cd = dbServicios.getResultadoComoArrayContenedorDatos(sql, false);
      for (int i = 0;i < cd.length;i++){
        System.out.println(cd[i]);
      }
    }catch(Exception e){
      e.printStackTrace();
      throw new RuntimeException(e);
    }finally{
      try{
        dbServicios.release(true);
      }catch(Exception ex){
        ex.printStackTrace();
      }
    }
  }

  public void clausulaIN(){
    System.out.println("\n======================== Ejemplo de uso de clausula 'IN' \n======================== usando
SELECT con numero de parametros \n======================== more than DBMS can except (like more then 254 in Oracle)
to populate objectList ==========================\n");
    try{
      dbServicios.setNumeroMaximoParametros(7);
      dbServicios.setMarcadorPosicion(":#");

      dbServicios.setAlmacenStatements(true);
      setRequestor(Cliente.class);
      setMapping(cargaMappings("Cliente.properties"));

      List argsList = new ArrayList();
```

```
        String sql = sqlMap.getProperty("clausulaIN");

        List custId = new ArrayList();
        custId.add(new Integer(0));
        custId.add(new Integer(1));
        custId.add(new Integer(2));
        custId.add(new Integer(3));
        custId.add(new Integer(4));
        custId.add(new Integer(5));
        custId.add(new Integer(6));
        custId.add(new Integer(7));

        argsList.add("Ricardo Garcia");

        argsList.add(custId);

        cargar(dbServicios.getResultSets(sql, argsList));

        Iterator it1 = getListaObjetos().iterator();
        while(it1.hasNext()){
           System.out.println(it1.next());
        }
        dbServicios.release(false);
        System.out.println("==================================");

        List list = dbServicios.getResultadoComoListaContenedorDatos(sql, argsList, false);

        Iterator it2 = list.iterator();
        while(it2.hasNext()){
           System.out.println(it2.next());
        }
     }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
     }finally{
        try{
           dbServicios.release(true);
        }catch(Exception ex){
           ex.printStackTrace();
        }
     }
  }

  public int buscaTodosClientes(){
     int records = 0;
     try{
        String sql = sqlMap.getProperty("buscaTodosClientes");
        records = this.cargar(this.dbServicios.getResultSet(sql));
        imprimeTodosClientes();
     }catch (SQLException sqle){
        throw new RuntimeException(sqle);
     }
     return records;
  }

  public int buscaTodosClientesMasTelefono(){
     System.out.println("\n======================= Ejemplo para uso de herencia ==========================\n");
     this.setRequestor(ClienteMasTelefono.class);

     setMapping(cargaMappings("ClienteMasTelefono.properties"));

     int records = 0;
     try{
        records = this.cargar(this.dbServicios.getResultSet(sqlMap.getProperty("buscaTodosClientesConTelefono")));
        imprimeTodosClientes();
     }catch (SQLException sqle){
        throw new RuntimeException(sqle);
     }
     return records;
  }
```

```
   public void creaAlmacenaClientes(){
      System.out.println("\n======================== Ejemplo de creacion y almacenaje de objetos (INSERT & UPDATE)
============================\n");
      this.setRequestor(ClienteMasTelefono.class);
      setMapping(cargaMappings("ClienteMasTelefono.properties"));
      this.getDBServicios().setAlmacenStatements(true);
      int records = buscaTodosClientes();
      ClienteMasTelefono cmt = new ClienteMasTelefono();

      cmt.setClienteId(new Integer(records + 1));
      cmt.setNombre("Ricardo");
      cmt.setApellidos("Garcia");
      cmt.setDireccion("Tomas Esteban, 17");
      cmt.setCiudad("Madrid");
      cmt.setTelefono("34914777387");

      this.crear(cmt);
      System.out.println("\ncreado cliente: " + cmt + "\n");

      ListIterator li = this.listaObjetos.listIterator();
      while (li.hasNext()){
         ClienteMasTelefono cliente = (ClienteMasTelefono) li.next();
         String apellidos = cliente.getApellidos();
         if (apellidos.equals("Gonzalez")){
            cliente.setTelefono("34934787351");
            cliente.almacenar();
         }
      }

      System.out.println("\n======================== Despues de las modificaciones de datos
============================\n");
      imprimeTodosClientes();

      try{
         this.dbServicios.beginTran();
         boolean success = this.grabarDatos();
         Throwable t = this.getErroresDetectados();
         if (t != null){
            throw t;
         }
         if (success){
            this.dbServicios.commitTran();
         }else{
            throw new SQLException("Modificacion datos: ha fallado");
         }
         this.sincronizarEstadoPersistencia();
      }catch (Throwable t){
         try{
            this.dbServicios.rollbackTran();
         }catch (SQLException sqle){
            sqle.printStackTrace();
            throw new RuntimeException(sqle);
         }
         t.printStackTrace();
      }finally{
         try{
            this.dbServicios.release(true);
         }catch (SQLException sqle){
            sqle.printStackTrace();
            throw new RuntimeException(sqle);
         }
      }

      System.out.println("\n================= Despues de hacer COMMIT & sincronizarEstadoPersistencia() o antes de
ROLLBACK ==================\n");
      imprimeTodosClientes();
   }

   public void buscaCliente(){
```

```
    System.out.println("\n====================== Ejemplo de uso de Criteria  ==========================\n");
    setMapping(cargaMappings("Cliente.properties"));
    this.rellenarMetadatos();
    List holders = new ArrayList(2);
    ContenedorCriteriosConsulta sCH1 = new ContenedorCriteriosConsulta();
    sCH1.setNombreCampo("apellidos");
    sCH1.setOperador(ContenedorCriteriosConsulta.LIKE);
    sCH1.setFunciones("UPPER(${})");
    sCH1.setBuscarValor("MI%");
    sCH1.setBooleanCondicion(ContenedorCriteriosConsulta.OR);
    holders.add(sCH1);
    ContenedorCriteriosConsulta sCH2 = new ContenedorCriteriosConsulta();
    sCH2.setNombreCampo("apellidos");
    sCH2.setOperador(ContenedorCriteriosConsulta.LIKE);
    sCH2.setFunciones("UPPER(${})");
    sCH2.setBuscarValor("S%");
    holders.add(sCH2);
    ClienteCriteria criteria = new ClienteCriteria();
    criteria.setMetadatos(this.metadatos);
    criteria.setContenedorCriteriosBusqueda(holders);
    criteria.construirCriterios();
    String select = this.sqlMap.getProperty("buscaTodosClientes");
    criteria.setSelect(select);
    String sql = criteria.getConsulta();
    Collection params = criteria.getArgumentos();

    try{
      this.dbServicios.setDebug(true);
      this.cargar(this.dbServicios.getResultSet(sql, params));
    }catch (SQLException sqle){
      throw new RuntimeException(sqle);
    }
    imprimeTodosClientes();
  }

  protected void imprimeTodosClientes(){
    List productos = this.getListaObjetos();
    Iterator iter = productos.iterator();
    while (iter.hasNext()){
      Cliente customer = (Cliente) iter.next();
      System.out.println(customer);
    }
  }
}
```

Tabla 9: Ejemplo de herencia para el empleo de FPUOC: FactoriaObjetoPersistente

6.6 Características adicionales de FPUOC

- En lugar de enviar todos los objetos por la red, en el caso de actualizaciones, es posible enviar solamente aquellos objetos modificados.
- También es posible generar sentencias DML sobre el cliente y enviarlos al mismo tiempo que los parámetros al servidor de aplicaciones o a otra aplicación para su procesamiento:

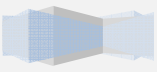
```
ClienteFactory cf = new ClienteFactory(Cliente.class);
cf.setGenerateSQLOnly(true);
...
cf.writeData();
...
List todosCambios = cf.getGeneradasSQL();

Una forma alternativa de realizar lo anterior es:

Object[] cambios = (Object[]) todosCambios.get(0);
String sql = (String) cambios[0];
Object arg = cambios[1];
PreparedStatement ps = connection.prepareStatement(sql, arg);
...

En el caso de que se utiliza FPUOC para las actualizaciones:

DBServicios dbServicios = new DBServicios();
...
dbServicios.ejecutar(sql, arg);
```

- Una forma muy sencilla de conseguir rellenar los objetos con los datos obtenidos de la SGBD es:

```
...
ClienteFactory cf = new ClienteFactory(Cliente.class);
ResultSet rs = ...;
cf.carga(rs);
...
```

- FPUOC mantiene la flexibilidad necesaria para que sea el usuario del framework quien escoja de qué manera se obtiene el resultado del la llamada a ResultSet: de otro framework concurrente, de sentencias SLQ directamente escritas por el usuario, etc. FPUOC no limita al usuario al uso del API predefinido de uso del mismo, sino que permite mucha versatilidad, en toda su extensión.
- No obstante, un método que puede ser usa es el siguiente:

```
...
ProductoFactory pf = new ProductoFactory(Producto.class);
List arg = new ArrayList(); arg.add(new Double(77.50));
arg.add("%CARD%");
String sql = "SELECT * FROM PRODUCTO WHERE PRECIO > ? AND NOMBRE LIKE ?";
DBServicios dbServicios = new DBServicios();
...
pf.carga(dbServicios.getResultSet(sql, arg));
dbServicios.release(true);
List productos = pf.getObjetoLista();
...
```

Generalmente se pueden almacenar las sentencias SQL en archivos de tipo .properties o bien en fichero .xml y añadir clausulas tipo WHERE basadas en las selecciones del usuario.

Hay varios objetos en el framework FPUOC concebidos (con metodos especiales para ello) para realizar esta actividad y facilitarla la labor.

- Ejecución en modo desconectado: se pueden rellanar los objetos de la BBDD o cualquier otra fuente, modificarlos, crear otros nuevos y luego serializarlos. Después se pueden restaurar, conectar la BBDDs o enviar los objetos a un servidor de aplicaciones y aplicar los cambios.
- Mantiene la capacidad de utilizar POJO para la persistencia (Plain Java Objects): solo es preciso extender (por herencia) desde la clase org.fpuoc.ObjetoPersistenteAbstracto, aunque no es la única manera.
- Se pueden generar actualizaciones (UPDATE) solamente para las columnas que han sido modificadas.
- Optimistic concurrency locking: la clausula WHERE para realizar actualización (UPDATE) y borrado (DELETE) se genera usando las propiedades:
  KEY_COLUMNS
  KEY_AND_MODIFIED_COLUMNS
  KEY_AND_UPDATEBLE_COLUMNS
- Secuencia para modificación de datos: cuando se modifican muchos objetos al tiempo, se puede especificar el orden de la modificación, por ejemplo se puede especificar que se realice primero un borrado (DELETE), luego una actualizacion (UPDATE), luego añadir datos (INSERT) (cualquier orden):

```
...
ProductoFactory pf = new ProductoFactory(Producto.class);
...
char[] dataModificacionSequence = new char[3];
dataModificacionSequence[0] = ProductoFactory.DATA_MODIFICATION_SEQUENCE_DELETE;
dataModificacionSequence[1] = ProductoFactory.DATA_MODIFICATION_SEQUENCE_UPDATE;
dataModificacionSequence[2] = ProductoFactory.DATA_MODIFICATION_SEQUENCE_INSERT;
pf.setDataModificacionSequence(dataModificacionSequence);
```

- Se pueden añadir los resultados de una sentencia SELECT a la lista de objetos en lugar de reemplazar el resultado previo, mediante el uso del método correspondiente, y se realiza mediante el uso de UNION ALL
- Soporte de paginación: se puede enviar al cliente o visualizar únicamente la cantidad específica de registros:

```
ProductoFactory pf = new ProductoFactory(Producto.class);
...
pf.carga(...
...
MarcadorPagina mp1 = this.getPagina(1);
System.out.println("\n================ Total registros: " + mp1.getObjetosTotales() + ", Pagina " + mp1.getNumeroPagina() +
" de " + mp1.getTotalPaginas() + " ================\n");

Iterator iter = mp1.getPagina().iterator();
while (iter1.hasNext()){
  Producto producto = (Producto) iter.next();
  System.out.println(producto);
}
...
```

- La arquitectura de FPUOC está basada en la aplicación del patrón DAO, así que si algún día se precisa cambiar a otro framework el uso de FPUOC no debe ser un inconveniente, y la conversión deberá ser relativamente simple.

7 FPUOC en ejecución

7.1 Presentación

Para la verificación del correcto funcionamiento del framework de persistencia FPUOC se presenta una sencilla colección de ejemplos de uso de los mismos.

Se ha tratado que la aplicación fuera lo más sencilla posible y al tiempo lo más completa para verificar en el mayor grado posible la flexibilidad, manejo y sencillez de codificación.

7.2 Características

El SGBD utilizado ha sido HSQLDB [http://hsqldb.org/], una sencilla BBDD, en su versión 1.8.0, que cumple los requerimientos mínimos para el ejemplo. Para la instalación según los requerimientos particulares, se remite a la documentación de la página web de referencia.

En el caso de este ejemplo, ya viene preconfigurada para su uso, y solo es preciso utilizar los lanzadores que acompañan la distribución.

Se ha utilizado este SGBD en vez de otros como ORACLE o MySQL debido a la sencillez de instalación y la flexibilidad en la portabilidad de los elementos de la BBDD y el uso de estándares JDBC usuales.

El entorno de desarrollo ha sido utilizando los componentes:

- NetBeans 6.1: utilizada para el desarrollo completo de todas las partes de la codificación y prueba. La entrega se realiza con el formato directo de uso en NetBeans, por lo que se puede importar la instalación y verificar la compilación y la codificación in situ.
- MagicDraw, versiones 11.5 y 15.5, para el desarrollo de los diagramas básicos de trabajo y arquitectura.
- Rational Rose Enterprise 2003, para la verificación de loa diagramas, y la confección de algunos diagramas de prueba adicionales.

En cuanto a la instalación de las pruebas, no se requiere más que seguir las instrucciones y el uso de los lanzadores (la aplicación solo está diseñada para su uso en entornos Windows y con java 1.5 o superior, preferiblemente la 6.0, para la que se han compilado en la versión final).

Los archivos .jar que se encuentran en el directorio ../lib son los únicos necesarios para la aplicación, junto a las extensión de la carpeta ../lib/ext.

7.3 El paquete de ejemplos

El paquete de ejemplos está diseñado para cargar un sencillo conjunto de datos (creando la BBDD en cada ejecución de los ejemplos) de forma que cada lanzamiento pueda verificar un uso de los métodos y objetos de la capa de persistencia FPUOC.

El código empleado es muy sencillo, utilizando un patrón factoria, y permite el la verificación de la generación de objetos para las tablas y el uso de los comando SQL y las respuestas esperadas.

Tal y como se puede verificar, no se necesitan fichero complejos de instalación, estilo .xml, para establecer la relación entre el framework y la BBDD. Únicamente se utilizan ficheros .properties, con las características iniciales de la BBDD y comando de uso, así como un fichero .xml, con el esquema DTD de empleo del mapeo de las SQL par la comprensión por el framework.

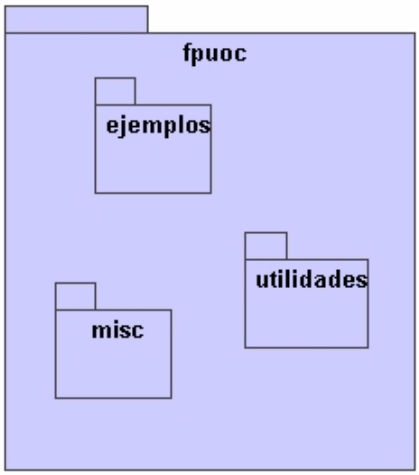


Figura 16: Esquema de paquetes de la entrega

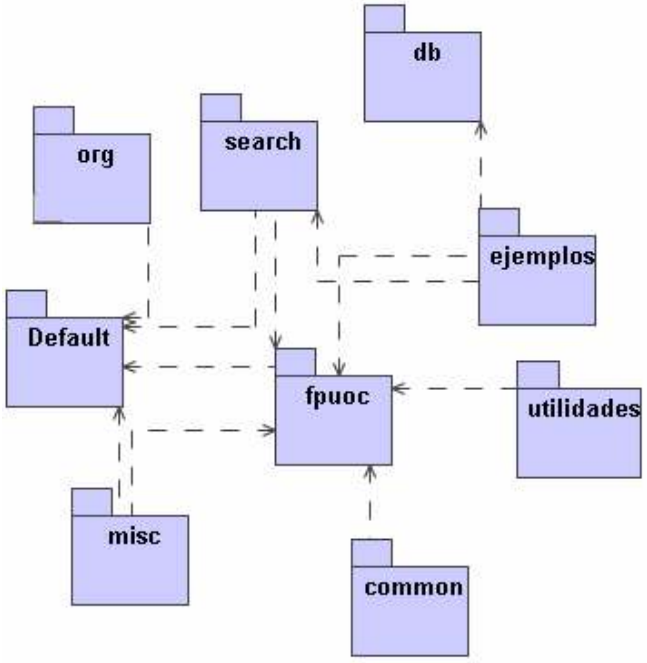


Figura 17: Dependencias de paquetes

7.4 Diagramas de la aplicación ejemplo

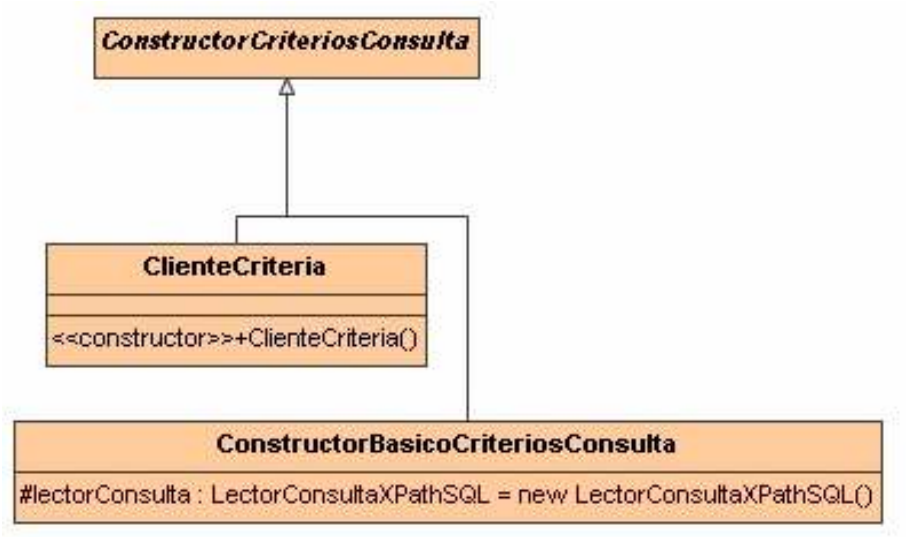


Figura 18: Jerarquía de las clases de consulta

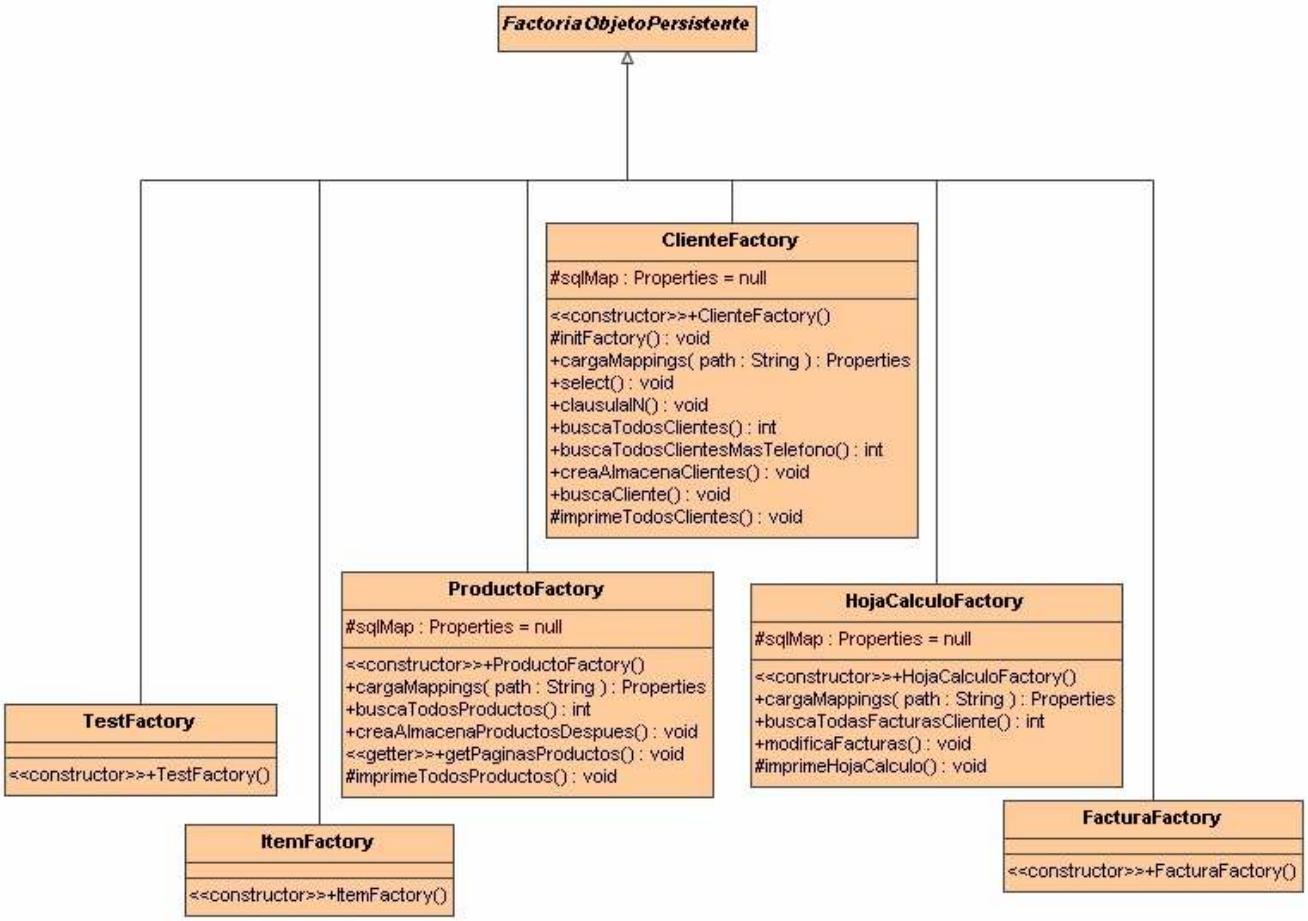


Figura 19: Jerarquía de los objetos factoría

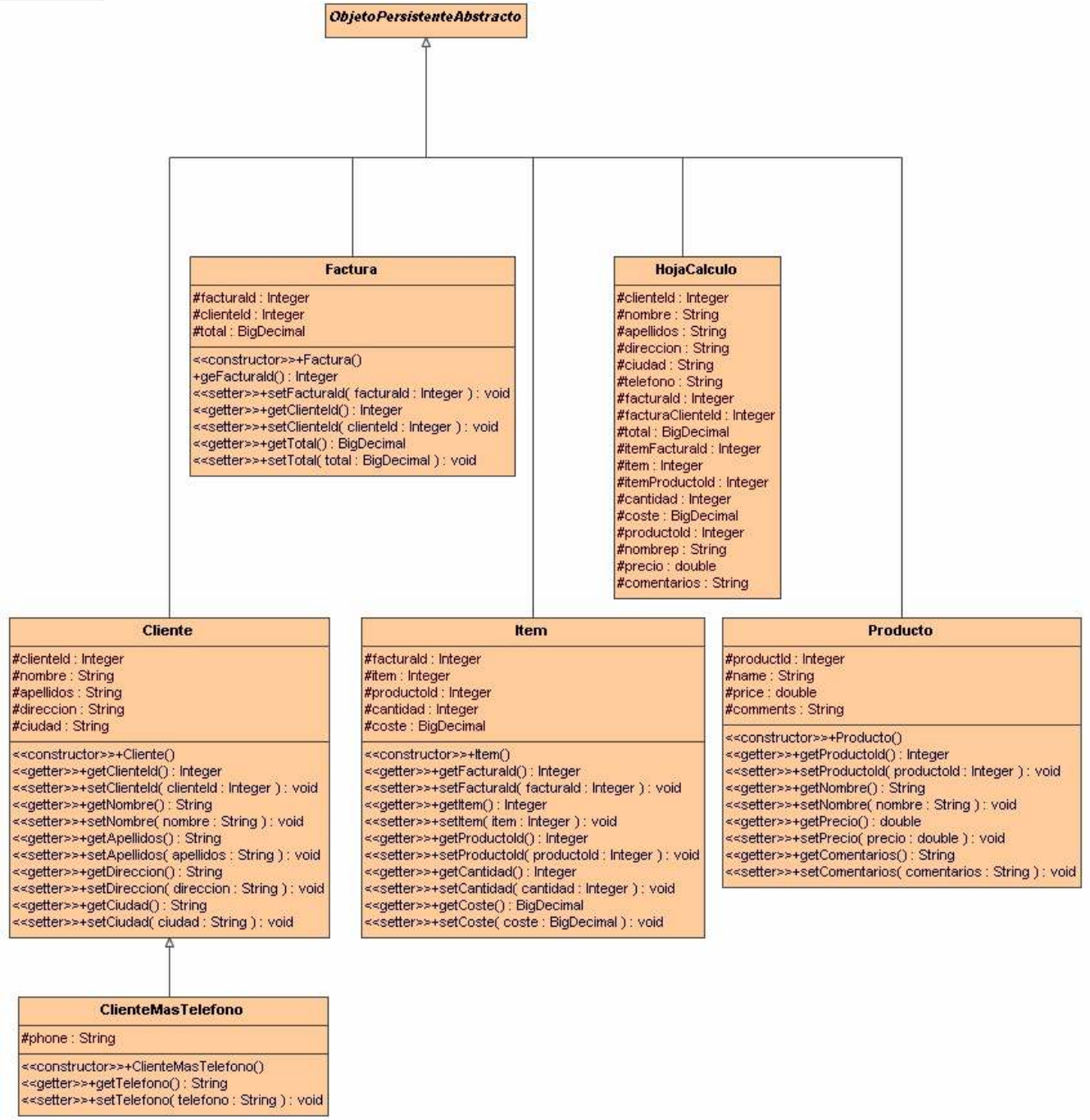


Figura 20: Jerarquía de los objetos de mapeo de las tablas

7.5 Instalación y uso de la aplicación de ejemplo

Para el uso de la aplicación las única condición es que esté instalado el kit de ejecución de Java, versión 5 o superior (preferiblemente la última ya que se ha compilado para la versión 1.6, aunque siempre está la opción de recompilar el código).

La aplicación viene con una estructura de directorios tanto para su instalación en el IDE NetBeans 6.1 como para su ejecución.

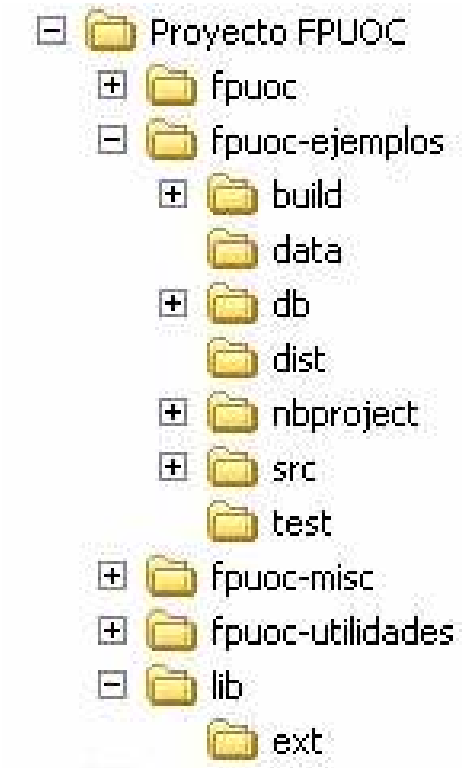


Figura 21: Estructura de directorios de la entrega

Cómo ejecutar los ejemplos

1. La aplicación solo se ha desarrollado para el entorno Windows, es decir, sus lanzadores de prueba (.bat)
2. Se debe ejecutar el servidor hsqldb en primer lugar (runServer.bat).
3. Se debe asegurar que el tamaño de la pantalla está ajustado para la visualización de la salida.
4. Ejecutar cada ejemplo. Cada no de los ejemplos, antes de ejecutarse, restaura la BBDD a su estado correcto (fpuoc_ejemplos/db/setup.sql).
5. De manera opcional, se puede ejecutar el GUI Database Manager (runManager.bat) para visualizar los datos almacenados.

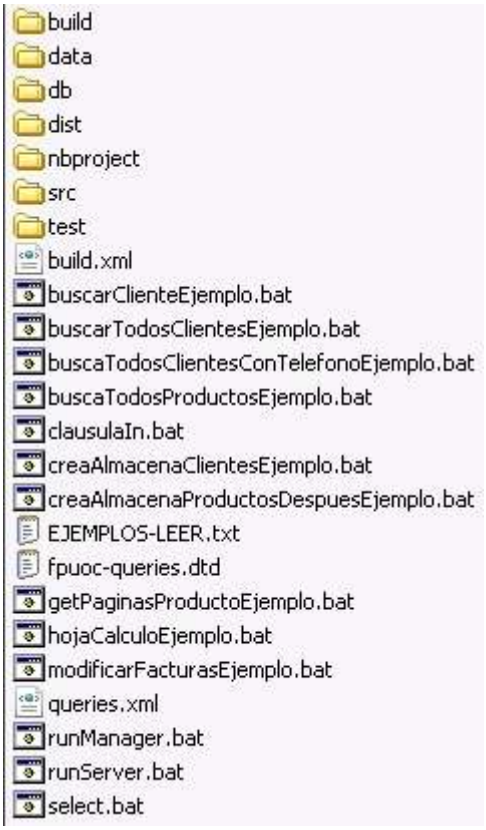
   Cuando se abra el Dialogo de Conexión seleccione HSQLDB Database Engine Server,

   en ningún caso debe usar HSQLDB Database Engine In-Memory o cualquier otro.

   Usuario: sa

   Password: <no tiene>

En la carpeta fpuoc-ejemplos están los ficheros necesarios para la prueba e indicados en la explicación anterior:



Glosario

- **Base de datos:** son la representación integra de los conjuntos de entidades o instancias de información y sus interrelaciones.
- **Caché:** memoria que permite mejorar el rendimiento del sistema al almacenar los datos más comunes y repetidos. De acuerdo composición es muy rápida, hecho que justifica su uso.
- **Capa:** también conocida como API (Interface de Programación de Aplicaciones). Es un conjunto de convenciones de programación que definen como se debe invocar un servicio desde un programa.
- **Ciclo de vida:** conjunto de etapas de un objeto o instancia.
- **Clase:** tipo de datos definido por el usuario que especifica un conjunto de objetas que comparten las mismas propiedades.
- **Clave de identificación:** Llave que garantiza la unicidad de las instancias o diferentes elementos.
- **Consulta:** cuestión realizada a una base de datos, en la que pide información o informaciones concretas en función de unos criterios definidos.
- **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. El término tiene un cierto paralelismo con el propio de las bases de datos.
- **Controlador:** también conocidos como controladores, son los encargados del trabajo Conjunto entre hardware y software.
- **Correspondencia:** término que en inglés traduciríamos como “mapping”, aun cuando su traducción literaria sería cartografía. Hace referencia al hecho que se produce cuando dos elementos son el mismo en aspecto pero de diferente naturaleza.
- **Encapsulación:** una de las 3 propiedades principales de los objetos. La encapsulación se define como la propiedad de los objetos que permite el acceso a su estado únicamente a través de su interfaz o de las relaciones preestablecidas con otros objetos.
- **Encriptación:** técnica por la que la información se hace ilegible a terceras personas. Para poder acceder a la información es siempre necesaria una llave que sólo conocen emisor y receptor.
- **Especificación:** es un documento que especifica cómo debe ser un sistema, un lenguaje, una capa, etc.
- **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 3 propiedades principales de los objetos. Permite generar clases y objetos a partir de otros ya existentes, manteniendo algunas o todas sus propiedades.
- **Incrustado:** en inglés ‘embedded’. Viene asociado al término lenguaje incrustado. Permite diferenciar un lenguaje que se encuentra dentro de otro.
- **Instancia:** El término instancia hace referencia siempre a un objeto creado por la clase en tiempo de ejecución conservando siempre las propiedades y métodos.
- **Implementación:** hace referencia a programar una aplicación a partir de un modelo de aplicación teórico o esquemático.
- **Java:** lenguaje de programación orientado a objetos, creado por Sun Microsystems pora generar aplicaciones exportables a Internet.
- **J2EE:** paquete del lenguaje Java que recoge sobre todo librerías orientadas a servidores de aplicaciones.
- **J2SE:** paquete del lenguaje Java que recoge las librerías básicas de este lenguaje.

- **Metadada:** término que define un documento que describe el contenido, la calidad, la condición y otras características un dato.
- **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:** caso concreto de una clase. Es la representación de un objeto real. Es producto de una Clase y cumple las propiedades requeridas del mismo objeto real que creemos que lo definen y que son necesarios por la nuestro 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.
- **PersistenceManager:** Agente administrador de la persistencia de los objetos.
- **PersistenceManagerFactory:** Administrador de todos los agentes de persistencia de cualquier aplicación.
- **Rollback:** literalmente quiere decir marcha atrás. Hace referencia al término propio de base de datos que nos permite recuperar el estado anterior a una operación sobre la información en una base de datos. No se debe confundir con el mismo término propio de las bases de datos.
- **Servidores de aplicaciones:** cualquier servidor que contiene la mayor parte de la lógica de una aplicación y la manipulación de datos. No contiene ninguna información en sí, sino el tratamiento y distribución de estas.
- **SGBDR:** nomenclatura que hace referencia a las bases de datos de tipo relacional.
- **SGBDOO:** nomenclatura que hace referencia a las bases de datos orientadas a objetos.
- **SQL:** lenguaje de consulta creado por IBM que facilita el tratamiento de datos de cualquier base de datos relacional.
- **TAD:** tipo abstracto de datos. Estructura de datos que permite representar tipos de datos avanzados.
- **Transacción:** es una acción. Mueve un dato de un lugar a otro. Se da entre bases de datos, de una aplicación a una base de datos, a la inversa, o bien entre aplicaciones. No se debe confundir con el término propio de las bases de datos.
- **Transparencia:** término que hace referencia al hecho que cualquier interacción entre dos aplicaciones, si depende de una tercera, estas trabajarán como si no existiera. Esto quiere decir que no resta rendimiento ni propiedades.
- **Unicidad:** una de las 3 propiedades principales de los objetos. Permite diferenciar como entidad un objeto en sí de otro de idénticas propiedades y valores en sus atributos.
- **XML:** también conocido como Extensible Markup Language, lenguaje extensible de marcas. Es un lenguaje dirigido al etiquetado de las partes que forman un documento y es además un metalenguaje que permite definir otros lenguajes, y que facilita la comunicación entre lenguajes de diferente naturaleza.

Bibliografía

M.ATKINSON R.MORRISON, Orthoganally Persistent Object Systems. The VLDB. Journal 4(3) pag 319-401. 1995

KENT BECK, Extreme Programming Explained, Embrace Change, Addison-Wesley Octubre, 1999

ELIZA BERTINO, LOREZON MARTINO. Sistemas de bases de datos orientados a objetos. Addison Wesley 1993

GRADY BOOCH, JAMES RUMBAUGH, IVAR JACOSON, El lenguaje Unificicado de Modelado, Adison Wesley,1999

BROWN, K. y WHITENACK, B. "Crossing Chasms: A Pattern Language for ObjectRDBMS Integration" dentro del Pattern Languages of Program Desing Vol 2, Addison-Wesley, 1996

R.G.G. CATTELL, Object Data Management Addison Wesley, 1994

C.J. DATE, Introducción a los sistemas de bases de datos, Addison-Wesley, 1986.

BRUCE ECKEL, Thinking Java. Prentice Hall.

ERICH GAMMA, RICHARD HELM, RALPH JONSON, JOHN VLISSIDES, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison Wesley, 1995

RON JEFFIRIES, ANN ANDERSEN, CHET HENDRICKSON, Extreme Programming Installed, Addison-Wesley Pub Co; Octubre, 2000

DAVID JORDAN, CRAIG RUSSELL Java Data Objects. O□Relly. April 2003

HENRY F.KORTH, ABRAHAM SILBERSCHATZ, Fundamentos de Bases de Datos, MgGraw Hill.

GRAIG LARMAN Uml y Patrones, Prentice Hall, 1999

BERTRAND MEYER, Object Oriented Software Construction 2nd Editon, Prentice Hall, 1997

NICHOLAS NEGROPONTE, Beeing Digital. Vintage Books. 1996

OMG Persistent State Service Specification 2.0, formal/02-09-06

R. ORFALI AND D. HARKEY, Client/Server Programming with Java and CORBA, John Wiley and Sons, New York, 1998. Client/Server Programming with Java and CORBA, 2nd Edition, Wiley, Marzo 1998 [24] CLAUS P. PRIESE, "A flexible Type-Extensible Object-Relational Database Wrapper- Architecture"

RIEL, ARTHUR J. Object-Oriented Design Heuristics. Addison-Wesley, 1996.

ROGER S. PRESSMAN, Sotware Engineering, McGraw Hill 3ed. 1991

JAMES R RUMBAUGH, MICHAEL R. BLAHA, WILLIAM LORENSEN FREDERICK EDDY , WILLIAM PREMERLANI, Object-Oriented Modeling and Desing, Prentice Hall.1990

SZYPERSKY, C. Component Software: Beyond Object-Oriented Programming. Addison -Wesley. Massachusetts. 1997.

SCOTT W. AMBLER Design of Robust Persistence Layer. http://www.ambysoft.com/persistenceLayer.html

DOUGLAS BARRY, DAVID JORDAN, ODMG: The Industry Standard for Java Object Storage http://www.objectidentity.com/images/componentstrategies.html

DOUGLAS BARRY, TORSTEN STANIENDA Solving the Java Object Storage Problem http://csdl.computer.org/comp/mags/co/1998/11/ry033abs.htm

DAVID JORDAN, Comparison Between Java Data Objects (JDO), Serialization an JDBC for Java Persistence http://www.jdocentral.com/pdf/DavidJordan_JDOversion_12Mar02.pdf

David Jordan, New Features in the JDO 2.0 Query Language http://jdocentral.com/JDO_Articles_20040329.html

DAVID JORDAN, The JDO Object Model, David Jordan, Java Report, 6/2001
http://www.objectidentity.com/images/jdo_model82.pdf

ARTHUR M. KELLER y otros, Architecting Object Applications for High Performance with Relational Databases, 1995 www.hep.net/chep95/html/papers/p59/p59.ps

WOLFGANG KELLER, Object/Relational Access Layers A Roadmap, missing Links and More Patterns http://www.objectarchitects.de/ObjectArchitects/orpatterns/

Productos de persistencia http://www.persistence.com/products/edgextend/index.php, http://www.objectmatter.com/, http://www.chimu.com/products/form/index.html, Cocobase(http://www.thoughtinc.com/cber_index.html), http://www.secant.com/Products/OI/index.html, Toplink(http://otn.oracle.com/products/ias/toplink/htdocs/sod.html) Hibernate (http://www.hibernate.org) ObjectRelationalBridge (OJB) http://db.apache.org/ojb/

Páginas Web de referencia

http://www.mail-archive.com/asnativos@5dms.com/msg10206.html
http://www.iscripting.net/smf/javascript/(js-framework)-create-your-own-mini-framework/0/
http://cc.borland.com/Item.aspx?id=22982
http://www.alistapart.com/articles/frameworksfordesigners
http://ammentos.biobytes.it/
http://www.codejava.org/detalle_nota.htm?idxnota=69479&destacada=1
http://www.codeproject.com/KB/architecture/WhatIsAFramework.aspx
http://www.javaworld.com/javaworld/jw-10-2005/jw-1003-mvc.html
http://web.fi.uba.ar/~dmontal/index.html
http://kuropka.net/
http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html
http://www.agiledata.org/essays/implementationStrategies.html
http://floggy.sourceforge.net/
http://www.cs.wustl.edu/~schmidt/rules.html
http://www.martincordova.com/
http://developer.apple.com/documentation/MacOSX/Conceptual/BPFrameworks/Frameworks.html
http://pegasus.javeriana.edu.co/~fwj2ee/
http://www.hibernate.org/
http://www.reviews.com/reviewer/quickreview/frameset_toplevel.cfm?bib_id=351360
http://nuestro.homelinux.org/clases_gal/Objetos/frameworks/
http://www.assembla.com/wiki/show/kumbia/Indice_wiki_kumbia
http://www.javaskyline.com/database.html
http://julp.sourceforge.net/
http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html
http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html?page=last
http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html
http://www.jdbaccess.com/
http://code.google.com/p/jlynx-persistence-framework/
http://www.juguy.org/content/view/157/1/
http://www.karvonite.com/
http://www.manning.com/
http://wiki.typo3.org/index.php/MVC_Framework
http://www.nabble.com/Floggy:-framework-para-persist%C3%AAncia-em-J2ME-MIDP-to12305612.html
http://st-www.cs.uiuc.edu/users/johnson/cs497/notes98/
http://wiki.typo3.org/index.php/Object_Persistence_Framework
http://www.dmoz.org/Computers/Programming/Languages/Java/Enterprise_Edition/Libraries_and_Frameworks/
http://java-source.net/open-source/persistence
http://www.google.com/search?ie=UTF-8&oe=UTF-8&gfns=1&sourceid=navclient&rls=com.google:es-ES:official&q=Open-Source+Web+Framework
http://dis.um.es/~jmolina/

http://code.google.com/p/persevere-framework/
http://www.matshelander.com/Weblog/DisplayLogEntry.aspx?LogEntryID=42
http://www.codeplex.com/Proetus
http://www.simplewebportal.net/host/1018.htm
http://soaagenda.com/journal/articulos/que-son-los-frameworks/
http://www.codeplex.com/queryframework
http://www.codeplex.com/spf
http://www.devx.com/Java/Article/33768/1954?pf=true
http://en.wikipedia.org/wiki/Software_framework
http://sourceforge.net/projects/ammentos
http://sourceforge.net/projects/gopf
http://www.ambysoft.com/essays/persistenceLayer.html
http://64.233.183.104/custom?q=cache:T_K1lqW6tq0J:dis.um.es/~jmolina/jisbd2000.ps+crear+un+framework+persistencia+en+java&hl=en&ct=clnk&cd=3&client=pub-5127604934005033
http://www.roseindia.net/enterprise/persistenceframework.shtml
http://diegumzone.spaces.live.com/?_c11_BlogPart_BlogPart=blogview&_c=BlogPart&partqs=amonth%3D6%26ayear%3D2006
http://www.ambysoft.com/scottAmbler.html

Anexos

Clase: ContenedorDatos.java

```
package org.fpuoc;

/**
 * Este objeto contiene los valores originales de la BBDD.
 * Estos valores se utilizan para desarrollar clausulas UPDATE y WHERE
 * con campos "1 - based" como en el caso de java.sql.ResultSet.
 */
public class ContenedorDatos implements java.io.Serializable, Cloneable {

   protected int contadorCampos = 0;
   protected Object[] datos = null;
   protected String[] nombreCampo = null;
   protected boolean ignorarNombreCampoIndefinido = false;

   public ContenedorDatos() {
   }

   public ContenedorDatos(int contadorCampos) {
      this.contadorCampos = contadorCampos;
      datos = new Object[contadorCampos];
   }

   public int getContadorCampos() {
      return contadorCampos;
   }

   public void setContadorCampos(int contadorCampos) {
      this.contadorCampos = contadorCampos;
   }

   public void setObjeto(int indiceCampos, Object valor) {
      this.datos[indiceCampos - 1] = valor;
   }

   public Object getObjeto(int indiceCampos) {
      return this.datos[indiceCampos - 1];
   }

   public void setNombreCampoYValor(int indiceCampo, String name, Object valor) {
      this.datos[indiceCampo - 1] = valor;
      if (this.nombreCampo == null) {
         this.nombreCampo = new String[this.contadorCampos];
      }
      this.nombreCampo[indiceCampo - 1] = name;
   }

   public void setObjeto(String nombre, Object valor) {
      int indiceCampo = findIndiceCampo(nombre);
      if (indiceCampo < 0) {
         if (this.nombreCampo == null) {
            if (!ignorarNombreCampoIndefinido) {
               throw new IllegalArgumentException("Nombre Campo no actualizado");
            }
            System.out.println("Nombre Campo no actualizado");
         } else {
            if (!ignorarNombreCampoIndefinido) {
               throw new IllegalArgumentException("No hay campo como: " + nombre);
            }
         }
      }
      this.datos[indiceCampo - 1] = valor;
   }

   public Object getObjeto(String nombre) {
      if (nombre == null) {
         return null;
      }
      int indiceCampo = findIndiceCampo(nombre);
```

```
    if (indiceCampo < 0) {
        if (this.nombreCampo == null) {
            if (!ignorarNombreCampoIndefinido) {
                throw new IllegalArgumentException("Nombre de Campo no actualizado");
            }
        } else {
            if (!ignorarNombreCampoIndefinido) {
                throw new IllegalArgumentException("No hay Campo como: " + nombre);
            }
        }
        return null;
    }
    return this.datos[indiceCampo - 1];
}

public void setNombreCampo(int indiceCampo, String nombre) {
    if (this.nombreCampo == null) {
        this.nombreCampo = new String[this.contadorCampos];
    }
    this.nombreCampo[indiceCampo - 1] = nombre;
}

public String getNombreCampo(int indiceCampo) {
    if (this.nombreCampo == null || this.nombreCampo.length == 0) {
        return null;
    }
    return this.nombreCampo[indiceCampo - 1];
}

public int findIndiceCampo(String nombre) {
    int indiceCampo = -1;
    if (this.nombreCampo == null || nombre == null) {
        return indiceCampo;
    }
    for (int i = 0; i < this.contadorCampos; i++) {
        if (nombreCampo[i] == null) {
            continue;
        }
        if (nombreCampo[i].equals(nombre)) {
            indiceCampo = i + 1;
            break;
        }
    }
    return indiceCampo;
}

@Override
public String toString() {
    StringBuffer sb = new StringBuffer();
    if (this.nombreCampo != null && this.nombreCampo.length == contadorCampos) {
        for (int i = 0; i < contadorCampos; i++) {
            sb.append(this.nombreCampo[i]).append("=");
            if (this.datos[i] == null) {
            } else {
                sb.append(this.datos[i]);
            }
            sb.append("&");
        }
        sb.deleteCharAt(sb.length() - 1);
    } else {
        for (int i = 0; i < contadorCampos; i++) {
            if (this.datos[i] == null) {
                sb.append("null");
            } else {
                sb.append(this.datos[i]);
            }
            sb.append(", ");
        }
        sb.deleteCharAt(sb.length() - 2);
    }
    return sb.toString();
}

public Object[] getDatos() {
    return this.datos;
}

public void setDatos(Object[] datos) {
    this.datos = datos;
}
```

```
  public boolean isIgnorarNombreCampoIndefinido() {
    return ignorarNombreCampoIndefinido;
  }

  public void setIgnorarNombreCampoIndefinido(boolean ignorarNombreCampoIndefinido) {
    this.ignorarNombreCampoIndefinido = ignorarNombreCampoIndefinido;
  }
}
```

Clase: DataSourceBasicoImpl.java

```
package org.fpuoc;

import javax.sql.DataSource;
import java.io.Serializable;
import java.sql.*;
import java.util.Properties;
import java.util.StringTokenizer;

public class DataSourceBasicoImpl implements DataSource, Serializable{

  protected transient Connection conexion;
  protected Driver driver;
  protected Properties propiedadesConexion = null;
  private String nombreDriver;
  private String dbURL;
  private String nombreUsuario;
  private String password;
  protected String conexionId = "";
  protected boolean debug = false;
  protected String delimitadorPropiedadesConexion = "&";

  /**
   * DataSourceBasicoImpl no tiene capacidad de crear pool
   * y debe ser usado para la prueba (test) o como cliente autonomo
   */

  public DataSourceBasicoImpl() {
    if (Boolean.getBoolean("debug-fpuoc") || Boolean.getBoolean("debug-" + getClass().getName())){
      debug = true;
    }
  }


  @Override
  protected DataSourceBasicoImpl clone() {
    DataSourceBasicoImpl bdsi = new DataSourceBasicoImpl();
    try{
      bdsi.setNombreUsuario(nombreUsuario);
      bdsi.setPassword(password);
      bdsi.setPropiedadesConexion(propiedadesConexion);
      bdsi.setDbURL(dbURL);
      bdsi.setDebug(debug);
      bdsi.setNombreDriver(nombreDriver);
      bdsi.setLogWriter(DriverManager.getLogWriter());
      bdsi.setLoginTimeout(DriverManager.getLoginTimeout());
    }catch(SQLException sqle){
      throw new RuntimeException(sqle);
    }
    return bdsi;
  }

  public java.sql.Connection getConnection() throws SQLException{
    if (conexion == null || conexion.isClosed()){

      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getConnection()::conexion
1: " + conexion + " \n");
      try {
        driver = (Driver) Class.forName(getNombreDriver()).newInstance();
        if (this.getPropiedadesConexion() == null || this.getPropiedadesConexion().isEmpty()){
          conexion = DriverManager.getConnection(getDbURL(), getNombreUsuario(), getPassword());
        }else{
```

```
                conexion = DriverManager.getConnection(getDbURL(), this.getPropiedadesConexion());
            }
        }catch (ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
            throw new RuntimeException(cnfe);
        }catch (SQLException sqle) {
            sqle.printStackTrace();
            throw new RuntimeException(sqle);
        }catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException(ex);
        }
    }
    if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getConnection()::conexion 2: "
+ conexion + " \n");
    return conexion;
  }

  @Override
  public Connection getConnection(String username, String password) throws SQLException {
    setNombreUsuario(username);
    setPassword(password);
    return getConnection();
  }

  @Override
  public java.io.PrintWriter getLogWriter() throws SQLException {
    return DriverManager.getLogWriter();
  }

  @Override
  public int getLoginTimeout() throws SQLException {
    return DriverManager.getLoginTimeout();
  }

  @Override
  public void setLogWriter(java.io.PrintWriter out) throws SQLException {
    DriverManager.setLogWriter(out);
  }

  @Override
  public void setLoginTimeout(int seconds) throws SQLException {
    DriverManager.setLoginTimeout(seconds);
  }

  public String getConexionId() {
    return conexionId;
  }

  public boolean isDebug() {
    return debug;
  }

  public void setDebug(boolean debug) {
    this.debug = debug;
  }

  public String getNombreDriver() {
    return this.nombreDriver;
  }

  public void setNombreDriver(String driverName) {
    this.nombreDriver = driverName;
  }

  public String getDbURL() {
    return dbURL;
  }

  public void setDbURL(String dbURL) {
    this.dbURL = dbURL;
  }

  public String getNombreUsuario() {
    return nombreUsuario;
  }

  public void setNombreUsuario(String userName) {
    this.nombreUsuario = userName;
  }
```

```
  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public void closeConexion() throws SQLException{
    if (conexion != null){
      conexion.close();
      conexion = null;
    }
  }

  public void setPropiedadesConexion(String propiedadesConexion) {
    StringTokenizer tokenizer = new StringTokenizer(propiedadesConexion, delimitadorPropiedadesConexion, false);
    if (this.propiedadesConexion == null){
      this.propiedadesConexion = new Properties();
    }else{
      this.propiedadesConexion.clear();
    }
    while (tokenizer.hasMoreTokens()){
      String token = tokenizer.nextToken();
      int idx = token.indexOf("=");
      if (idx == -1){
        throw new IllegalArgumentException("DataSourceBasicoImpl::setPropiedadesConexion(String propiedadesConexion): " +
propiedadesConexion + "(Formato argumento: <NOMBRE_PROPIEDAD>=<VALOR_PROPIEDAD>[DELIMITADOR]...)");
      }
      this.propiedadesConexion.put(token.substring(0, idx).trim(), token.substring(idx + 1).trim());
    }
  }

  public Properties getPropiedadesConexion() {
    return propiedadesConexion;
  }

  public void setPropiedadesConexion(Properties propiedadesConexion) {
    this.propiedadesConexion = propiedadesConexion;
  }

  public String getDelimitadorPropiedadesConexion() {
    return delimitadorPropiedadesConexion;
  }

  public void setDelimitadorPropiedadesConexion(String delimitadorPropiedadesConexion) {
    this.delimitadorPropiedadesConexion = delimitadorPropiedadesConexion;
  }

  @Override
  public boolean isWrapperFor(Class iface){
    return false;
  }

  @Override
  public Class unwrap(Class iface){
    return null;
  }
}
```

Clase: DBGrabarDatos.java

```
package org.fpuoc;

import java.util.*;
import java.sql.*;
import java.beans.*;
import java.lang.reflect.*;

public class DBGrabarDatos implements GrabarDatos, java.io.Serializable, Cloneable {

  public DBGrabarDatos() {
    if (Boolean.getBoolean("debug-fpuoc") || Boolean.getBoolean("debug-" + getClass().getName())) {
```

```
        setDebug(true);
      }
    }
    protected int contadorBajas = 0;
    protected int contadorAltas = 0;
    protected int contadorModificaciones = 0;
    protected List objetosEliminados = null;
    protected List objetosCreados = null;
    protected List objetosModificados = null;
    protected List generadasSQL = null;
    protected FactoriaObjetoPersistente peticion = null;
    protected String nombreTablaCompleto = null;
    protected String modificacionCatalogo = null;
    protected String modificacionEsquema = null;
    protected String tablaModificada = null;
    protected Object[] ARG_LECT_VACIO = new Object[0];
    protected boolean debug = false;
    protected Throwable erroresDetectados = null;
    protected boolean excepcionEnObjectListVacio = false;

    protected void init() throws SQLException {
      contadorBajas = 0;
      contadorAltas = 0;
      contadorModificaciones = 0;
      if (peticion.getMetadatos() == null) {
        peticion.rellenarMetadatos();
      }
      modificacionCatalogo = this.peticion.getCatalogo();
      modificacionEsquema = this.peticion.getEsquema();
      tablaModificada = this.peticion.getTabla();
      erroresDetectados = null;
      if (objetosEliminados == null) {
        objetosEliminados = new ArrayList();
      }
      if (objetosCreados == null) {
        objetosCreados = new ArrayList();
      }
      if (objetosModificados == null) {
        objetosModificados = new ArrayList();
      }
      if (tablaModificada == null) {
        throw new SQLException("Nombre de tabla no existe");
      }
      nombreTablaCompleto = tablaModificada;
      if (modificacionEsquema != null) {
        nombreTablaCompleto = modificacionEsquema + "." + nombreTablaCompleto;
        if (modificacionCatalogo != null) {
          nombreTablaCompleto = modificacionCatalogo + "." + nombreTablaCompleto;
        }
      }
    }

    @Override
    public boolean grabaDatos(FactoriaObjetoPersistente peticion) {
      if (peticion.isSoloLectura()) {
        setErroresDetectados(new SQLException("Solo lectura"));
        return false;
      }
      this.peticion = peticion;

      try {
        this.init();
      } catch (SQLException sqle) {
        setErroresDetectados(sqle);
        return false;
      }
      boolean completado = true;
      boolean vacio = false;
      if (debug) {
        System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::writeData()::peticion.getObjectList()
=================== " + peticion.getListaObjetos());
      }
      if (peticion.getListaObjetos() == null || peticion.getListaObjetos().isEmpty()) {
        vacio = true;
      }
      objetosEliminados = peticion.getObjetosEliminados();
      if (debug) {
        System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::writeData()::objetosEliminados
=================== " + objetosEliminados);
      }
      if ((objetosEliminados == null || objetosEliminados.isEmpty()) && vacio) {
```

```
        if (excepcionEnObjectListVacio) {
          setErroresDetectados(new SQLException("No hay nada para grabar"));
          return false;
        }
        return true;
      }
      if (peticion.getDatosModificacionSequence() != null) {
        Iterator it = peticion.getListaObjetos().iterator();
        while (it.hasNext()) {
          ObjetoPersistente objetoPersistente = (ObjetoPersistente) it.next();
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::writeData()::objetoPersistente
=================== " + objetoPersistente);
          }
          if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.ELIMINADO && objetoPersistente.getValoresOriginales() != null &&
objetoPersistente.getValoresOriginales().getContadorCampos() > 0) {
            objetosEliminados.add(objetoPersistente);
          } else if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.ALMACENADO && objetoPersistente.getValoresOriginales() !=
null && objetoPersistente.getValoresOriginales().getContadorCampos() > 0) {
            objetosModificados.add(objetoPersistente);
          } else if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.CREADO && objetoPersistente.getValoresOriginales() == null ||
objetoPersistente.getValoresOriginales().getContadorCampos() < 1) {
            objetosCreados.add(objetoPersistente);
          }
        }
        for (int i = 0; i < peticion.getDatosModificacionSequence().length; i++) {
          if (peticion.getDatosModificacionSequence(i) == FactoriaObjetoPersistente.DATOS_MODIFICACION_SEQUENCE_DELETE) {
            eliminaObjetos();
          } else if (peticion.getDatosModificacionSequence(i) == FactoriaObjetoPersistente.DATOS_MODIFICACION_SEQUENCE_UPDATE) {
            almacenaObjetos();
          } else if (peticion.getDatosModificacionSequence(i) == FactoriaObjetoPersistente.DATOS_MODIFICACION_SEQUENCE_INSERT) {
            creaObjetos();
          }
        }
      } else {
        peticion.getListaObjetos().addAll(objetosEliminados);
        Iterator it = peticion.getListaObjetos().iterator();
        while (it.hasNext()) {
          ObjetoPersistente objetoPersistente = (ObjetoPersistente) it.next();
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::writeData()::objetoPersistente
=================== " + objetoPersistente);
          }
          if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.ELIMINADO && objetoPersistente.getValoresOriginales() != null &&
objetoPersistente.getValoresOriginales().getContadorCampos() > 0) {
            if (!eliminaObjeto(objetoPersistente)) {
              return false;
            }
          } else if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.ALMACENADO && objetoPersistente.getValoresOriginales() !=
null && objetoPersistente.getValoresOriginales().getContadorCampos() > 0) {
            if (!almacenarObjeto(objetoPersistente)) {
              return false;
            }
          } else if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.CREADO && objetoPersistente.getValoresOriginales() == null ||
objetoPersistente.getValoresOriginales().getContadorCampos() < 1) {
            if (!creaObjeto(objetoPersistente)) {
              return false;
            }
          } else if (objetoPersistente.getEstadoPersistencia() == ObjetoPersistente.ORIGINAL) {
            // no hace nada
          } else {
            setErroresDetectados(new SQLException("Invalido PersistentState"));
            return false;
          }
        }
      }
      return completado;
    }

    protected boolean eliminaObjetos() {
      if (objetosEliminados.isEmpty()) {
        return true;
      }
      Iterator iter = objetosEliminados.iterator();
      while (iter.hasNext()) {
        ObjetoPersistente objetoPersistente = (ObjetoPersistente) iter.next();
        if (!eliminaObjeto(objetoPersistente)) {
          return false;
        }
```

```
    }
    return true;
  }

  protected boolean almacenaObjetos() {
    Iterator iter = objetosModificados.iterator();
    while (iter.hasNext()) {
      ObjetoPersistente objetoPersistente = (ObjetoPersistente) iter.next();
      if (!almacenarObjeto(objetoPersistente)) {
        return false;
      }
    }
    return true;
  }

  protected boolean creaObjetos() {
    Iterator iter = objetosCreados.iterator();
    while (iter.hasNext()) {
      ObjetoPersistente objetoPersistente = (ObjetoPersistente) iter.next();
      if (!creaObjeto(objetoPersistente)) {
        return false;
      }
    }
    return true;
  }

  @SuppressWarnings("static-access")
  protected boolean eliminaObjeto(ObjetoPersistente objetoPersistente) {
    String nombreCampo = null;
    Collection colParm = new ArrayList();
    StringBuffer sql = new StringBuffer("DELETE FROM ");
    try {
      sql.append(nombreTablaCompleto);
      sql.append(" WHERE ");
      /* Siempre se deben usar columnas con PrimaryKeys en clausulas WHERE */
      List pk = peticion.getPrimaryKey(modificacionCatalogo, modificacionEsquema, tablaModificada);
      Iterator iter = pk.iterator();
      while (iter.hasNext()) {
        String nombreColumna = (String) iter.next();   // nombre de columna completo

        nombreCampo = (String) peticion.getMapping().get(nombreColumna);
        int idx = peticion.getMetadatos().getIndiceColumnaPorNombreCampo(nombreCampo);
        Method leerMethod = peticion.getMetadatos().getLeerMethod(idx);
        Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
        if (valorOriginal == null) {
          sql.append(nombreColumna).append(" IS NULL AND ");
        } else {
          sql.append(nombreColumna).append(" = ? AND ");
          colParm.add(valorOriginal);
        }
      }
      if (peticion.getOptimisticLock() == FactoriaObjetoPersistente.KEY_COLUMNS) { /* built de nuevo */

        /* Para DELETE se deben poner iguales KEY_AND_MODIFIED_COLUMNS y KEY_AND_UPDATEBLE_COLUMNS: se debe incluir todas las
columnas de tabla en la cláusula WHERE */
      } else if (peticion.getOptimisticLock() == peticion.KEY_AND_MODIFIED_COLUMNS || peticion.getOptimisticLock() ==
FactoriaObjetoPersistente.KEY_AND_UPDATEBLE_COLUMNS) {
        for (int i = 1; i <= peticion.getMetadatos().getContadorColumna(); i++) {
          String catalogo = peticion.getMetadatos().getNombreCatalogo(i);
          String esquema = peticion.getMetadatos().getNombreEsquema(i);
          String tabla = peticion.getMetadatos().getNombreTabla(i);
          if (modificacionCatalogo != null) {
            if (!modificacionCatalogo.equals(catalogo)) {
              continue;
            }
          }
          if (modificacionEsquema != null) {
            if (!modificacionEsquema.equals(esquema)) {
              continue;
            }
          }
          if (tablaModificada != null) {
            if (!tablaModificada.equals(tabla)) {
              continue;
            }
          }
          String nombreColumna = peticion.getMetadatos().getCompletoNombreColumna(i);
          if (sql.indexOf(nombreColumna) > -1) {
            continue;// esta es una columna tipo PK
          }
          if (!peticion.getMetadatos().isGrabable(i) || peticion.getMetadatos().isSoloLectura(i)) {
```

```
                continue;
              }
              nombreCampo = peticion.getMetadatos().getNombreCampo(i);
              try {
                Object valor = leerValor(objetoPersistente, peticion.getMetadatos().getLeerMethod(i));
                Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
                if (valorOriginal == null) {
                  sql.append(nombreColumna).append(" IS NULL AND ");
                } else {
                  sql.append(nombreColumna).append(" = ? AND ");
                  colParm.add(valorOriginal);
                }
              } catch (Throwable t) {
                throw new SQLException(t.getMessage());
              }
            }
          } else {
            throw new SQLException("Ajuste para bloqueo optimo (Optimistic Lock) invalido");
          }
          int sqlLong = sql.length();
          sql = sql.delete(sqlLong - 5, sqlLong - 1);
          if (peticion.isGenerarSoloSQL()) {
            if (generadasSQL == null) {
              generadasSQL = new ArrayList();
            }
            Object[] nota = new Object[2];
            nota[0] = sql.toString();
            nota[1] = colParm.toArray();
            generadasSQL.add(nota);
          } else {
            int filasAfectadas = peticion.getDBServicios().ejecutar(sql.toString(), colParm);
            if (peticion.isThrowOptimisticLockDeleteException() && filasAfectadas != 1) {
              return false;
            }
            setContadorBajas(contadorBajas + filasAfectadas);
          }
        } catch (SQLException sqlEx) {
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::eliminaObjeto::Excepcion in
eliminaObjeto: " + sqlEx);
          }
          setErroresDetectados(sqlEx);
          sqlEx.printStackTrace();
          return manejadorExcepcion(objetoPersistente, sql.toString(), colParm, sqlEx);
        }
        return true;
      }

      @SuppressWarnings("static-access")
      protected boolean almacenarObjeto(ObjetoPersistente objetoPersistente) {
        String nombreCampo = null;
        StringBuffer sql = new StringBuffer("UPDATE ");
        List listParms = new ArrayList();
        if (debug) {
          System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::almacenarObjeto
====================");
        }
        try {
          sql.append(nombreTablaCompleto).append(" SET ");
          for (int i = 1; i <= peticion.getMetadatos().getContadorColumna(); i++) {
            String catalogo = peticion.getMetadatos().getNombreCatalogo(i);
            String esquema = peticion.getMetadatos().getNombreEsquema(i);
            String tabla = peticion.getMetadatos().getNombreTabla(i);
            if (modificacionCatalogo != null) {
              if (!modificacionCatalogo.equals(catalogo)) {
                continue;
              }
            }
            if (modificacionEsquema != null) {
              if (!modificacionEsquema.equals(esquema)) {
                continue;
              }
            }
            if (tablaModificada != null) {
              if (!tablaModificada.equals(tabla)) {
                continue;
              }
            }
            if (!peticion.getMetadatos().isGrabable(i) || peticion.getMetadatos().isSoloLectura(i)) {
```

```
            continue;
          }
          String nombreColumna = null;
          if (peticion.isNoNombreColumnaCompleto()) {
            nombreColumna = peticion.getMetadatos().getNombreColumna(i);
          } else {
            nombreColumna = peticion.getMetadatos().getCompletoNombreColumna(i);
          }
          nombreCampo = peticion.getMetadatos().getNombreCampo(i);
          Object valor = leerValor(objetoPersistente, peticion.getMetadatos().getLeerMethod(i));
          Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +
"::almacenarObjeto::nombreCampo: " + nombreCampo + " value: " + valor + " origValue: " + valorOriginal + " isWritable: " +
peticion.getMetadatos().isGrabable(i) + " isReadOnly: " + peticion.getMetadatos().isSoloLectura(i) + " ==================");
          }
          if (valorOriginal == null) {
            if (valor == null) {
              continue; // mismo valor - no se debe incluir en la actualización (update)
            } else {
              sql.append(nombreColumna).append(" = ?, ");
              listParms.add(valor);
            }
          } else {
            if (valor == null) {
              sql.append(nombreColumna).append(" = NULL, ");
            } else {
              if (!valorOriginal.equals(valor)) {
                sql.append(nombreColumna).append(" = ?, ");
                listParms.add(valor);
              } else {
                continue;
              }
            }
          }
        }
        int idx = sql.lastIndexOf(", ");
        if (idx > -1) {
          sql = sql.delete(idx, idx + 2);
        } else { // datos de ninguna columna modificada
          objetoPersistente.setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
          return true;
        }
        /* Construir WHERE */
        /* Siempre se deben usar columnas tipo PrimaryKeys en clausulas WHERE */
        StringBuffer where = new StringBuffer(" WHERE ");
        List pk = peticion.getPrimaryKey(modificacionCatalogo, modificacionEsquema, tablaModificada);
        if (debug) {
          System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::almacenarObjeto::pk: " + pk + "
modificacionEsquema: " + modificacionEsquema + " tablaModificada: " + tablaModificada);
        }
        Iterator iter = pk.iterator();
        while (iter.hasNext()) {
          String nombreColumna = (String) iter.next();
          nombreCampo = (String) peticion.getMapping().get(nombreColumna);
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +
"::almacenarObjeto::nombreColumna: " + nombreColumna + " nombreCampo: " + nombreCampo);
          }
          int index = peticion.getMetadatos().getIndiceColumnaPorNombreCampo(nombreCampo);
          Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
          if (valorOriginal == null) {
            where.append(nombreColumna).append(" IS NULL AND ");
          } else {
            where.append(nombreColumna).append(" = ? AND ");
            listParms.add(valorOriginal);
          }
        }
        if (peticion.getOptimisticLock() == FactoriaObjetoPersistente.KEY_COLUMNS) {
          idx = where.lastIndexOf(" AND ");
          where = where.delete(idx + 1, idx + 5);
        }
        if (peticion.getOptimisticLock() == FactoriaObjetoPersistente.KEY_COLUMNS) {
          // ... construir de nuevo
        } else if (peticion.getOptimisticLock() == FactoriaObjetoPersistente.KEY_AND_MODIFIED_COLUMNS) {
          for (int i = 1; i <= peticion.getMetadatos().getContadorColumna(); i++) {
            String catalogo = peticion.getMetadatos().getNombreCatalogo(i);
            String esquema = peticion.getMetadatos().getNombreEsquema(i);
            String tabla = peticion.getMetadatos().getNombreTabla(i);
            if (modificacionCatalogo != null) {
              if (!modificacionCatalogo.equals(catalogo)) {
```

```
                continue;
            }
        }
        if (modificacionEsquema != null) {
            if (!modificacionEsquema.equals(esquema)) {
                continue;
            }
        }
        if (tablaModificada != null) {
            if (!tablaModificada.equals(tabla)) {
                continue;
            }
        }
        String nombreColumna = peticion.getMetadatos().getCompletoNombreColumna(i);
        if (where.indexOf(nombreColumna) > -1) {
            continue;// esta es una columna tipo PK

        }
        if (!peticion.getMetadatos().isGrabable(i) || peticion.getMetadatos().isSoloLectura(i)) {
            continue;
        }
        nombreCampo = peticion.getMetadatos().getNombreCampo(i);
        Object valor = leerValor(objetoPersistente, peticion.getMetadatos().getLeerMethod(i));
        Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
        if (valorOriginal == null) {
            if (valor == null) {
                continue;
            } else {
                where.append(nombreColumna).append(" IS NULL AND ");
            }
        } else {
            if (valor == null) {
                where.append(nombreColumna).append(" = ? AND ");
                listParms.add(valorOriginal);
            } else {
                if (valorOriginal.equals(valor)) {
                    continue;
                } else {
                    where.append(nombreColumna).append(" = ? AND ");
                    listParms.add(valorOriginal);
                }
            }
        }
    }
} else if (peticion.getOptimisticLock() == FactoriaObjetoPersistente.KEY_AND_UPDATEBLE_COLUMNS) {
    for (int i = 1; i <= peticion.getMetadatos().getContadorColumna(); i++) {
        String catalogo = peticion.getMetadatos().getNombreCatalogo(i);
        String esquema = peticion.getMetadatos().getNombreEsquema(i);
        String tabla = peticion.getMetadatos().getNombreTabla(i);
        if (modificacionCatalogo != null) {
            if (!modificacionCatalogo.equals(catalogo)) {
                continue;
            }
        }
        if (modificacionEsquema != null) {
            if (!modificacionEsquema.equals(esquema)) {
                continue;
            }
        }
        if (tablaModificada != null) {
            if (!tablaModificada.equals(tabla)) {
                continue;
            }
        }
        String nombreColumna = peticion.getMetadatos().getCompletoNombreColumna(i);
        if (where.indexOf(nombreColumna) > -1) {
            continue;    // esta es una columna tipo PK

        }
        if (!peticion.getMetadatos().isGrabable(i) || peticion.getMetadatos().isSoloLectura(i)) {
            continue;
        }
        nombreCampo = peticion.getMetadatos().getNombreCampo(i);
        Object valor = leerValor(objetoPersistente, peticion.getMetadatos().getLeerMethod(i));
        Object valorOriginal = objetoPersistente.getValorOriginal(nombreCampo);
        if (valorOriginal == null) {
            where.append(nombreColumna).append(" IS NULL AND ");
        } else {
            where.append(nombreColumna).append(" = ? AND ");
```

```
                listParms.add(valorOriginal);
              }
            }
          } else {
            throw new SQLException("Ajuste para bloqueo optimo (Optimistic Lock) invalido");
          }
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::almacenarObjeto::where 1 : " +
where);
          }
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +
"::almacenarObjeto::peticion.KEY_COLUMNS 1 : " + peticion.KEY_COLUMNS);
          }
          if (peticion.getOptimisticLock() != FactoriaObjetoPersistente.KEY_COLUMNS) {
            idx = where.length();
            where = where.delete(idx - 5, idx - 1);
          }
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::almacenarObjeto::where 2 : " +
where);
          }
          sql.append(where);
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +
"::almacenarObjeto::almacenarObjeto: " + sql + "; listParms: " + listParms);
          }
          if (peticion.isGenerarSoloSQL()) {
            if (generadasSQL == null) {
              generadasSQL = new ArrayList();
            }
            Object[] nota = new Object[2];
            nota[0] = sql.toString();
            nota[1] = listParms.toArray();
            generadasSQL.add(nota);
          } else {
            int filasAfectadas = peticion.getDBServicios().ejecutar(sql.toString(), listParms);
            if (peticion.isThrowOptimisticLockUpdateException() && filasAfectadas != 1) {
              if (debug) {
                System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +
"::almacenarObjeto::filasAfectadas: " + filasAfectadas);
              }
              return false;
            }
            setContadorModificaciones(contadorModificaciones + filasAfectadas);
          }
        } catch (SQLException sqlEx) {
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::almacenarObjeto::Excepcion en
almacenarObjeto: " + sqlEx);
          }
          setErroresDetectados(sqlEx);
          sqlEx.printStackTrace();
          return manejadorExcepcion(objetoPersistente, sql.toString(), listParms, sqlEx);
        }
        return true;
      }

      protected boolean creaObjeto(ObjetoPersistente objetoPersistente) {
        String nombreCampo = null;
        StringBuffer sql = new StringBuffer("INSERT INTO ");
        List listParms = new ArrayList();
        try {
          sql.append(nombreTablaCompleto);
          sql.append(" (");
          for (int i = 1; i <= peticion.getMetadatos().getContadorColumna(); i++) {
            String catalogo = peticion.getMetadatos().getNombreCatalogo(i);
            String esquema = peticion.getMetadatos().getNombreEsquema(i);
            String tabla = peticion.getMetadatos().getNombreTabla(i);
            if (modificacionCatalogo != null) {
              if (!modificacionCatalogo.equals(catalogo)) {
                continue;
              }
            }
            if (modificacionEsquema != null) {
              if (!modificacionEsquema.equals(esquema)) {
                continue;
              }
            }
            if (tablaModificada != null) {
              if (!tablaModificada.equals(tabla)) {
```

```
                continue;
              }
            }
            String nombreCampoTemp = peticion.getMetadatos().getNombreCampo(i);
            if (!peticion.getMetadatos().isGrabable(i) || peticion.getMetadatos().isSoloLectura(i)) {
              continue;
            }
            String nombreColumna = null;
            if (peticion.isNoNombreColumnaCompleto()) {
              nombreColumna = peticion.getMetadatos().getNombreColumna(i);
            } else {
              nombreColumna = peticion.getMetadatos().getCompletoNombreColumna(i);
            }
            sql.append(nombreColumna).append(", ");
            Object valor = leerValor(objetoPersistente, peticion.getMetadatos().getLeerMethod(i));
            listParms.add(valor);
          }
          int idx = sql.lastIndexOf(", ");
          sql = sql.delete(idx, idx + 2);
          sql.append(") VALUES (");
          ListIterator li = listParms.listIterator();
          while (li.hasNext()) {
            Object valor = li.next();
            if (valor == null) {
              sql.append("NULL, ");
              li.remove();
            } else {
              sql.append("?, ");
            }
          }
          idx = sql.lastIndexOf(",");
          sql = sql.delete(idx, idx + 2);
          sql.append(")");
          if (peticion.isGenerarSoloSQL()) {
            if (generadasSQL == null) {
              generadasSQL = new ArrayList();
            }
            Object[] nota = new Object[2];
            nota[0] = sql.toString();
            nota[1] = listParms.toArray();
            generadasSQL.add(nota);
          } else {
            int filasAfectadas = peticion.getDBServicios().ejecutar(sql.toString(), listParms);
            if (peticion.isThrowFailedInsertException() && filasAfectadas != 1) {
              throw new SQLException("INSERT INTO " + nombreTablaCompleto + " ha fallado");
            }
            setContadorAltas(contadorAltas + filasAfectadas);
          }
        } catch (SQLException sqlEx) {
          if (debug) {
            System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this + "::creaObjeto::Excepcion en
creaObjeto: " + sqlEx);
          }
          setErroresDetectados(sqlEx);
          sqlEx.printStackTrace();
          return manejadorExcepcion(objetoPersistente, sql.toString(), listParms, sqlEx);
        }
        return true;
      }

      /** Leer la propiedad de contadorBajas.
       * @return Value of property contadorBajas.
       *
       */
      @Override
      public int getContadorBajas() {
        return contadorBajas;
      }

      /** Asignar nueva propiedad a contadorBajas.
       * @param contadorBajas New valor of property contadorBajas.
       *
       */
      protected void setContadorBajas(int contadorBajas) {
        this.contadorBajas = contadorBajas;
      }

      /** Leer la propiedad de contadorAltas.
       * @return Value of property contadorAltas.
```

```
 *
 */
@Override
public int getContadorAltas() {
   return contadorAltas;
}

/** Asignar nueva propiedad a contadorAltas.
 * @param contadorAltas New valor of property contadorAltas.
 *
 */
protected void setContadorAltas(int contadorAltas) {
   this.contadorAltas = contadorAltas;
}

/** Leer la propiedad de contadorModificaciones.
 * @return Value of property contadorModificaciones.
 *
 */
@Override
public int getContadorModificaciones() {
   return contadorModificaciones;
}

/** Asignar nueva propiedad a contadorModificaciones.
 * @param contadorModificaciones New valor of property contadorModificaciones.
 *
 */
protected void setContadorModificaciones(int contadorModificaciones) {
   this.contadorModificaciones = contadorModificaciones;
}

@Override
public void reset() {
   contadorBajas = 0;
   contadorAltas = 0;
   contadorModificaciones = 0;
   generadasSQL = null;
   objetosEliminados = null;
   objetosCreados = null;
   objetosModificados = null;
   nombreTablaCompleto = null;
   tablaModificada = null;
   modificacionEsquema = null;
   modificacionCatalogo = null;
   erroresDetectados = null;
}

/** Leer la propiedad de generadasSQL.
 * @return Value of property generadasSQL.
 *
 */
@Override
public List getSQLGeneradas() {
   return generadasSQL;
}

/** Leer la propiedad de erroresDetectados.
 * @return Value of property erroresDetectados.
 *
 */
@Override
public Throwable getErroresDetectados() {
   return erroresDetectados;
}

/** Asignar nueva propiedad a erroresDetectados.
 * @param erroresDetectados New valor of property erroresDetectados.
 *
 */
public void setErroresDetectados(Throwable erroresDetectados) {
   this.erroresDetectados = erroresDetectados;
}

/** Leer la propiedad de modificacionCatalogo.
 * @return Value of property modificacionCatalogo.
 *
 */
public String getModificacionCatalogo() {
   return modificacionCatalogo;
}
```

```
/** Asignar nueva propiedad a modificacionCatalogo.
 * @param modificacionCatalogo New valor of property modificacionCatalogo.
 *
 */
public void setModificacionCatalogo(String modificacionCatalogo) {
   this.modificacionCatalogo = modificacionCatalogo;
}

/** Leer la propiedad de generatedSQLasXML.
 * @return Value of property generatedSQLasXML.
 */
@Override
public String getGenerarSentenciasSQLcomoXML() {
   if (this.generadasSQL == null || this.generadasSQL.isEmpty()) {
      return "";
   }
   StringBuffer sb = new StringBuffer("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\n<generated-sql>\n");
   Iterator it = this.generadasSQL.iterator();
   while (it.hasNext()) {
      sb.append(" <sql>\n");
      Object[] objeto = (Object[]) it.next();
      sb.append("  <statement><![CDATA[").append(objeto[0]).append("]]></statement>\n");
      Object[] parametros = (Object[]) objeto[1];
      if (parametros != null && parametros.length > 0) {
         sb.append("  <params>\n");
         for (int i = 0; i < parametros.length; i++) {
            sb.append("   <param>\n");
            sb.append("    <value>").append("<![CDATA[").append(parametros[i].toString()).append("]]>").append("</value>\n");
            sb.append("    <datatype>").append(parametros[i].getClass().getName()).append("</datatype>\n");
            sb.append("   </param>\n");
         }
         sb.append("  </params>\n");
      }
      sb.append(" </sql>\n");
   }
   sb.append("</generated-sql>\n");
   return sb.toString();
}

protected boolean manejadorExcepcion(ObjetoPersistente objetoPersistente, String sql, Collection colParms, Throwable t) {
   boolean ignorar = false;
   if (t instanceof StringIndexOutOfBoundsException) {
      System.out.println("Asegurese de usar correctametne el esquema (schema) y/o la tabla");
   }
   if (peticion.getManejadorExcepciones() != null) {
      ignorar = peticion.getManejadorExcepciones().manejadorExcepcion(objetoPersistente, sql, colParms, t);
   }
   return ignorar;
}

public boolean isDebug() {
   return debug;
}

public void setDebug(boolean debug) {
   this.debug = debug;
}

@Override
public boolean isExcepcionEnObjectListVacios() {
   return excepcionEnObjectListVacio;
}

@Override
public void setExcepcionEnObjectListVacios(boolean excepcionEnObjectListVacio) {
   this.excepcionEnObjectListVacio = excepcionEnObjectListVacio;
}

protected Object leerValor(ObjetoPersistente objetoPersistente, Method metodo) throws SQLException {
   Object valor = null;
   try {
      valor = metodo.invoke(objetoPersistente, ARG_LECT_VACIO);
   } catch (Throwable t) {
      if (t instanceof InvocationTargetException) {
         throw new RuntimeException(((InvocationTargetException) t).getTargetException());
      } else {
         throw new RuntimeException(t);
      }
```

```
      }
      return valor;
    }
}
```

Clase: DBServicios.java

```
package org.fpuoc;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import javax.naming.*;
import javax.sql.*;
import java.sql.*;
import java.util.*;

public class DBServicios extends Object implements java.io.Serializable, Cloneable {

    public static final char NONE = 'N';
    public static final char ROLLBACK = 'R';
    public static final char COMMIT = 'C';

    protected boolean conexionCerrada = false;
    protected transient Context context;
    protected transient DataSource dataSource;
    protected transient Connection connection;
    protected transient Statement statement;
    protected transient PreparedStatement preparedStatement;
    protected transient CallableStatement callStatement = null;
    protected transient ResultSet resultSet;

    protected Properties conexionProperties = null;
    protected String nombreOrigenDatos = null;
    protected String nombreCursor = null;
    protected int maxLongitudCampo = 0;
    protected int maxFilas = 0;
    protected int queryTiempoLimite = 0;
    protected int fetchDireccion = 0;
    protected int fetchLongitud = 0;
    //TYPE_FORWARD_ONLY = 1003, TYPE_SCROLL_INSENSITIVE = 1004, TYPE_SCROLL_SENSITIVE = 1005
    protected int resultadoSetType = ResultSet.TYPE_FORWARD_ONLY;
    //CONCUR_READ_ONLY, CONCUR_UPDATABLE
    protected int concurrencia = ResultSet.CONCUR_READ_ONLY;
    protected boolean soloLectura = false;
    protected boolean cerrarStatementTrasEjecucion = true;
    protected boolean escapeProcessing = true;
    protected boolean transaccionEnCurso = false;
    protected boolean resetOrigenDatos = false;
    protected boolean debug = false;
    protected boolean advertenciasProcesadas = false;
    protected boolean proccessStoredProc = false;
    protected boolean cerrarResultSet = true;
    protected char proccessTransactionOnCloseConnection = NONE;
    protected int maxFilasProcesarTransaccion = 0; // cuantas filas para to update/insert/delete en la transacción
    protected int filasAfectadasPorTransaccion = 0;
    protected String conexionId = "<none>";
    protected String marcadorPosicion = null;
    protected int numeroMaximoParametros = -1;
    protected transient List resultSets = null;
    protected List advertencias = null;
    //indicadores de cómo manejar ResultSet(s) desde Stored Procedures y Functions
    public static final int IGNORAR_RESULTADO = 0; // no procesar ResultSet(s)
    public static final int RESULTADO_LISTA_CONTENEDOR_DATOS = 1;
    public static final int RESULTADO_VECTOR = 2; // Vector de vectores
    public static final int RESULTADO_MAP = 3; // Map of Lists
    public static final int RESULTADO_ROWSET = 4; // CachedRowSetImpl

    protected boolean almacenStatements = false;
    protected int maxLongitudAlmacenamientoSentencias = 20; // se debe porner antes del inicio de sentenciasAlmacenamiento, por defecto: 20
    protected List sentenciasId; // id de sentencias almacenadas (sql String)
    protected List sentenciasAlmacenamiento;  // sentencias almacenadas
    protected List contadorSentenciasAlmacenadas; // contador de uso de sentencias almacenadas
    protected String cachedRowSetClassName = "com.sun.rowset.CachedRowSetImpl"; // se asigna otra implementacion...
    /**
     * Si EJBContext no es nulo indica que este es el EJB Container que maneja
```

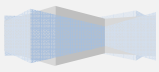
```
     * la transacción y en lugar de connection.rollback() se hará
     * EJBContexts.setRollbackOnly y no connection.setAutoCommit (false),
     * ni tampoco connection.commit().
     */
    protected Object EJBContext = null;

    public DBServicios() {
        if (Boolean.getBoolean("debug-fpuoc") || Boolean.getBoolean("debug-" + getClass().getName())){
            debug = true;
        }
    }

    /** Leer la propiedad de context.
     * @return Value of property context.
     *
     */
    public javax.naming.Context getContext(){
        if (context == null){
            try{
                context = new InitialContext();
                if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getContext()::context: " +
context + " \n");
            }catch (NamingException ne){
                throw new RuntimeException(ne);
            }
        }
        return context;
    }

    /** Asignar nueva propiedad a context.
     * @param context New valor of property context.
     *
     */
    public void setContext(javax.naming.Context context) {
        this.context = context;
    }

    /** Leer la propiedad de connection.
     * @return Value of property connection.
     *
     */
    public java.sql.Connection getConnection()  throws SQLException{
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getConnection()::connection
1: " + connection + "::isTransaccionEnCurso(): " + transaccionEnCurso + " \n");
        if (isTransaccionEnCurso()){
            return this.connection;
        }
        if (connection == null || connection.isClosed()){

            dataSource = this.getDataSource();
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::getConnection()::dataSource 1: " + dataSource + " \n");
            connection = dataSource.getConnection();

            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getConnection()::connection
2: " + connection + " \n");
        }
        if (isTransaccionEnCurso()){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::getConnection()::isTransaccionEnCurso(): " + isTransaccionEnCurso() + "::conexionId: " + conexionId + "\n");
            if (!this.conexionId.equals(connection.toString())){
                throw new SQLException("Se cerro la conexion mientras se ejecutaba una transaccion");
            }
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(connection.getWarnings());
            connection.clearWarnings();
        }
        return connection;
    }

    /** Asignar nueva propiedad a connection.
     * @param connection New valor of property connection.
     *
     */
    public void setConnection(java.sql.Connection connection) {
        this.connection = connection;
    }
```

```
   /** Leer la propiedad de dataSource.
    * @return Value of property dataSource.
    *
    */
   public javax.sql.DataSource getDataSource() throws SQLException{
      try{
         if (dataSource == null || isResetOrigenDatos()){
            if (nombreOrigenDatos == null){
               throw new SQLException("Nombre de DataSource no existe");
            }else if (nombreOrigenDatos.indexOf("/") == -1){
               dataSource = (DataSource) this.getContext().lookup("java:comp/env/jdbc/" + nombreOrigenDatos);
            }else{
               dataSource = (DataSource) this.getContext().lookup(nombreOrigenDatos);
            }
         }
      }catch (NamingException ne){
         throw new RuntimeException(ne.getMessage());
      }
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getDataSource()::dataSource:
" + dataSource + " \n");
      return dataSource;
   }

   /** Asignar nueva propiedad a dataSource.
    * @param dataSource New valor of property dataSource.
    *
    */
   public void setDataSource(javax.sql.DataSource dataSource) {
      this.dataSource = dataSource;
   }

   protected void prepararStatement(String sql) throws SQLException{
      this.preparedStatement = this.getConnection().prepareStatement(sql, getResultadoSetType(), getConcurrencia());
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::prepararStatement()::sql: \n"
+ sql + " \n");
      this.preparedStatement.setEscapeProcessing(this.getEscapeProcessing());
      this.preparedStatement.setFetchSize(this.getFetchLongitud());
      this.preparedStatement.setMaxRows(this.getMaxFilas());
      this.preparedStatement.setMaxFieldSize(this.getMaxLongitudCampo());
      this.preparedStatement.setQueryTimeout(this.getQueryTiempoLimite());
      if (this.getNombreCursor() != null){
         this.preparedStatement.setCursorName(this.getNombreCursor());
      }
   }

   /** Leer la propiedad de prepararStatement.
    * @return Value of property prepararStatement.
    *
    */
   public PreparedStatement getPreparedStatement(String sql, Collection colParms) throws SQLException{
      if (sql == null || sql.trim().length() == 0) {
         throw new SQLException("SQL String esta en blanco o es null");
      }
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
getPreparedStatement(): \n" + sql + " ::colParms: " + colParms + " \n");
      int idx = -1;
      if (almacenStatements){  // PreparedStatement será usado nuevamente, si no debe cerrarse y asignarle = null
         idx = sentenciasId.indexOf(sql);
         if (idx >= 0){
            Object objeto = sentenciasAlmacenamiento.get(idx);
            if (debug) {
               System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
getPreparedStatement() ============ hay almacenadas preparedStatement: \n" + "idx: " + idx + " " + sql + "::" + (objeto ==
null?"<null>":objeto.toString() + objeto.getClass()) + " \n");
               System.out.println("fpuoc ============ sentenciasId: " + sentenciasId);
               System.out.println("fpuoc ============ sentenciasAlmacenamiento: " + sentenciasAlmacenamiento);
               System.out.println("fpuoc ============ contadorSentenciasAlmacenadas: " + contadorSentenciasAlmacenadas);
            }
            preparedStatement = (PreparedStatement) sentenciasAlmacenamiento.get(idx);
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
getPreparedStatement() ============ hay almacenadas preparedStatement: \n" + sql + "::" + preparedStatement + " \n");
            if (preparedStatement == null){
               sentenciasId.remove(idx);
               sentenciasAlmacenamiento.remove(idx);
               idx = -1;
            }else{
               preparedStatement.clearParameters();
            }
         }
      }
      if (idx == -1){
```

```
            prepararStatement(sql);
        }
        Iterator paramsIter = colParms.iterator();
        int parmIdx = 1;
        while(paramsIter.hasNext()){
            Object param = paramsIter.next();
            if (param != null){
                if (param.getClass().getName().equals("Date")){
                    param = new java.sql.Date(((java.util.Date) param).getTime());
                }
                preparedStatement.setObject(parmIdx, param);
            }else{
                preparedStatement.setNull(parmIdx, Types.JAVA_OBJECT);
            }
            parmIdx++;
        }
        if (almacenStatements){
            if (idx == -1){
                idx = addAlmacenStatement(sql, preparedStatement);
                contadorSentenciasAlmacenadas.add(new Integer(1));
            }else{
                // ¿Cuantas veces se ha llamado a prepararStatement?
                Object contador = contadorSentenciasAlmacenadas.get(idx);
                if (contador == null){
                    contadorSentenciasAlmacenadas.add(new Integer(1));
                }else{
                    contadorSentenciasAlmacenadas.set(idx, new Integer(((Integer) contador).intValue() + 1));
                }
            }
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(preparedStatement.getWarnings());
            preparedStatement.clearWarnings();
        }
        return preparedStatement;
    }

    /** Asignar nueva propiedad a prepararStatement.
     * @param preparedStatement New valor of property prepararStatement.
     *
     */
    public void setPreparedStatement(java.sql.PreparedStatement preparedStatement) {
        this.preparedStatement = preparedStatement;
    }

    /**********************************************************************/

    public ContenedorDatos[] getResultadoComoArrayContenedorDatos(String sql, Collection colParms, boolean llenarNombresColumna) throws
SQLException{
        return (ContenedorDatos[]) getResultadoComoListaContenedorDatos(sql, colParms, llenarNombresColumna).toArray(new
ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadoComoArrayContenedorDatos(String sql, boolean llenarNombresColumna) throws SQLException{
        return (ContenedorDatos[]) getResultadoComoListaContenedorDatos(sql, llenarNombresColumna).toArray(new ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadoComoArrayContenedorDatos(String sql) throws SQLException{
        return (ContenedorDatos[]) getResultadoComoListaContenedorDatos(sql).toArray(new ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadoComoArrayContenedorDatos(String sql, Collection colParms) throws SQLException{
        return (ContenedorDatos[]) getResultadoComoListaContenedorDatos(sql, colParms).toArray(new ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadosComoArrayContenedorDatos(String sql, Collection colParms) throws SQLException{
        return (ContenedorDatos[]) getResultadosComoListaContenedorDatos(sql, colParms, true).toArray(new ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadosComoArrayContenedorDatos(String sql, Collection colParms, boolean llenarNombresColumna) throws
SQLException{
        return (ContenedorDatos[]) getResultadosComoListaContenedorDatos(sql, colParms, llenarNombresColumna).toArray(new
ContenedorDatos[0]);
    }

    public ContenedorDatos[] getResultadosComoArrayContenedorDatos(String sql, Collection colParms, int maxNumeroParametros, String
marcadorPosicion) throws SQLException{
        return (ContenedorDatos[]) getResultadosComoListaContenedorDatos(sql, colParms, maxNumeroParametros, marcadorPosicion,
true).toArray(new ContenedorDatos[0]);
```

```
    }

    public ContenedorDatos[] getResultadosComoArrayContenedorDatos(String sql, Collection colParms, int maxNumeroParametros, String
marcadorPosicion, boolean llenarNombresColumna) throws SQLException{
        return (ContenedorDatos[]) getResultadosComoListaContenedorDatos(sql, colParms, maxNumeroParametros, marcadorPosicion,
llenarNombresColumna).toArray(new ContenedorDatos[0]);
    }

    /*************************************************************************/

    public Object getValorUnico(String sql) throws SQLException{
        Object valor = null;
        getResultSet(sql);
        if(this.resultSet.next()){
            valor = this.resultSet.getObject(1);
        }else{
            throw new SQLException("No data");
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return valor;
    }

    public Object getValorUnico(String sql, Collection colParms) throws SQLException{
        Object valor = null;
        getResultSet(sql, colParms);
        if(this.resultSet.next()){
            valor = this.resultSet.getObject(1);
        }else{
            throw new SQLException("No data");
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return valor;
    }

    /*************************************************************************/

    public List getResultadoUnicaColumnaComoLista(String sql) throws SQLException{
        getResultSet(sql);
        List lista = new ArrayList();
        while(this.resultSet.next()){
            lista.add(this.resultSet.getObject(1));
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return lista;
    }

    public List getResultadoUnicaColumnaComoLista(String sql, Collection colParms) throws SQLException{
        getResultSet(sql, colParms);
        List lista = new ArrayList();
        while(this.resultSet.next()){
            lista.add(this.resultSet.getObject(1));
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return lista;
    }

    public List getResultadosUnicaColumnaComoLista(String sql, Collection colParms, int maxNumeroParametros, String marcadorPosicion) throws
SQLException{
        getResultSets(sql, colParms, maxNumeroParametros, marcadorPosicion);
        List lista = new ArrayList();
        for (int i = 0;i < resultSets.size();i++){
            while (((ResultSet) resultSets.get(i)).next() ){
                lista.add(((ResultSet) resultSets.get(i)).getObject(1));
            }
            if (isAdvertenciasProcesadas()){
                rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
                ((ResultSet) resultSets.get(i)).clearWarnings();
            }
        }
```

```
    return lista;
  }

  public List getResultadosUnicaColumnaComoLista(String sql, List listParms) throws SQLException{
    getResultSets(sql, listParms);
    List lista = new ArrayList();
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        lista.add(((ResultSet) resultSets.get(i)).getObject(1));
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    return lista;
  }

  /*****************************************************************************/

  public Map getResultadoDosColumnasComoMap(String sql) throws SQLException{
    getResultSet(sql);
    Map colMap = new LinkedHashMap();
    while(this.resultSet.next()){
      colMap.put(this.resultSet.getObject(1), this.resultSet.getObject(2));
    }
    if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(this.resultSet.getWarnings());
      this.resultSet.clearWarnings();
    }
    return colMap;
  }

  public Map getResultadoDosColumnasComoMap(String sql, Collection colParms) throws SQLException{
    getResultSet(sql, colParms);
    Map colMap = new LinkedHashMap();
    while(this.resultSet.next()){
      colMap.put(this.resultSet.getObject(1), this.resultSet.getObject(2));
    }
    return colMap;
  }

  public Map getResultadosDosColumnasComoMap(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion)
throws SQLException{
    getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
    Map colMap = new LinkedHashMap();
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        colMap.put(this.resultSet.getObject(1), this.resultSet.getObject(2));
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    return colMap;
  }

  public Map getResultadosDosColumnasComoMap(String sql, List listParams) throws SQLException{
    getResultSets(sql, listParams);
    Map colMap = new LinkedHashMap();
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        colMap.put(this.resultSet.getObject(1), this.resultSet.getObject(2));
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    return colMap;
  }

  /*****************************************************************************/

  public List getResultadoComoValoresListaObjetos(String sql) throws SQLException{
    getResultSet(sql);
    List listObj = new ArrayList();
    while(this.resultSet.next()){
```

```
            listObj.add(new ValorObjeto(this.resultSet.getObject(1), this.resultSet.getString(2)));
        }
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return listObj;
    }

    public List getResultadoComoValoresListaObjetos(String sql, Collection colParms) throws SQLException{
        getResultSet(sql, colParms);
        List listObj = new ArrayList();
        while(this.resultSet.next()){
            listObj.add(new ValorObjeto(this.resultSet.getObject(1), this.resultSet.getString(2)));
        }
        return listObj;
    }

    public List getResultadoComoValoresListaObjetos(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion)
throws SQLException{
        getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
        List listObj = new ArrayList();
        for (int i = 0;i < resultSets.size();i++){
            while (((ResultSet) resultSets.get(i)).next() ){
                listObj.add(new ValorObjeto(this.resultSet.getObject(1), this.resultSet.getString(2)));
            }
            if (isAdvertenciasProcesadas()){
                rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
                ((ResultSet) resultSets.get(i)).clearWarnings();
            }
        }
        return listObj;
    }

    public List getResultadosComoValoresListaObjetos(String sql, List params) throws SQLException{
        getResultSets(sql, params);
        List listObj = new ArrayList();
        for (int i = 0;i < resultSets.size();i++){
            while (((ResultSet) resultSets.get(i)).next() ){
                listObj.add(new ValorObjeto(this.resultSet.getObject(1), this.resultSet.getString(2)));
            }
            if (isAdvertenciasProcesadas()){
                rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
                ((ResultSet) resultSets.get(i)).clearWarnings();
            }
        }
        return listObj;
    }

    /*****************************************************************************/

    public List getResultadoComoListaContenedorDatos(String sql, Collection colParms, boolean rellenarNombresColumnas) throws SQLException{
        getResultSet(sql, colParms);
        ResultSetMetaData rsmd = this.resultSet.getMetaData();
        int contColum = rsmd.getColumnCount();
        List listDatos = new ArrayList();
        String[] nombresColum = new String[contColum];
        Map dups = new HashMap();
        if (rellenarNombresColumnas){
            // rename mapDatos if there are duplicate names
            for (int col = 1;col <= contColum;col++){
                String nombreColum = rsmd.getColumnName(col);
                if (!dups.containsKey(nombreColum)){
                    dups.put(nombreColum, new Integer(0));
                }else{
                    Integer num = (Integer) dups.get(nombreColum);
                    int intValor = num.intValue();
                    num = new Integer(intValor + 1);
                    dups.put(nombreColum, num);
                    nombreColum = nombreColum + num;
                }
                nombresColum[col - 1] = nombreColum;
            }
        }
        while(this.resultSet.next()){
            ContenedorDatos cd = new ContenedorDatos(contColum);
            for (int col = 1;col <= contColum;col++){
                if (rellenarNombresColumnas){
                    cd.setNombreCampoYValor(col, nombresColum[col - 1], this.resultSet.getObject(col));
                }else{
                    cd.setObjeto(col, this.resultSet.getObject(col));
```

```
            }
          }
          listDatos.add(cd);
        }
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(this.resultSet.getWarnings());
          this.resultSet.clearWarnings();
        }
        return listDatos;
    }

    public List getResultadoComoListaContenedorDatos(String sql, boolean rellenarNombresColumnas) throws SQLException{
        getResultSet(sql);
        ResultSetMetaData rsmd = this.resultSet.getMetaData();
        int contColum = rsmd.getColumnCount();
        List listDatos = new ArrayList();
        String[] nombresColum = new String[contColum];
        Map dups = new HashMap();
        if (rellenarNombresColumnas){
          // rename mapDatos if there are duplicate names
          for (int col = 1;col <= contColum;col++){
            String nombreColum = rsmd.getColumnName(col);
            if (!dups.containsKey(nombreColum)){
              dups.put(nombreColum, new Integer(0));
            }else{
              Integer num = (Integer) dups.get(nombreColum);
              int intValor = num.intValue();
              num = new Integer(intValor + 1);
              dups.put(nombreColum, num);
              nombreColum = nombreColum + num;
            }
            nombresColum[col - 1] = nombreColum;
          }
        }
        while(this.resultSet.next()){
          ContenedorDatos cd = new ContenedorDatos(contColum);
          for (int col = 1;col <= contColum;col++){
            if (rellenarNombresColumnas){
              cd.setNombreCampoYValor(col, nombresColum[col - 1], this.resultSet.getObject(col));
            }else{
              cd.setObjeto(col, this.resultSet.getObject(col));
            }
          }
          listDatos.add(cd);
        }
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(this.resultSet.getWarnings());
          this.resultSet.clearWarnings();
        }
        return listDatos;
    }

    public List getResultadoComoListaContenedorDatos(String sql) throws SQLException{
        return getResultadoComoListaContenedorDatos(sql, true);
    }

    public List getResultadoComoListaContenedorDatos(String sql, Collection colParms) throws SQLException{
        return getResultadoComoListaContenedorDatos(sql, colParms, true);
    }

    public List getResultadosComoListaContenedorDatos(String sql, Collection colParms) throws SQLException{
        return getResultadosComoListaContenedorDatos(sql, colParms, true);
    }

    public List getResultadosComoListaContenedorDatos(String sql, Collection params, int numeroMaximoParametros, String marcadorPosicion)
throws SQLException{
        return getResultadosComoListaContenedorDatos(sql, params, numeroMaximoParametros, marcadorPosicion, true);
    }

    public List getResultadosComoListaContenedorDatos(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion,
boolean rellenarNombresColumnas) throws SQLException{
        getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
        ResultSetMetaData rsmd = ((ResultSet) this.resultSets.get(0)).getMetaData();
        int contColum = rsmd.getColumnCount();
        List listDatos = new ArrayList();
        String[] nombresColum = new String[contColum];
        Map dups = new HashMap();
        // cambiar el nombre a  mapDatos si hay nombres duplicados
        for (int col = 1;col <= contColum;col++){
```

```
        String nombreColum = rsmd.getColumnName(col);
        if (!dups.containsKey(nombreColum)){
          dups.put(nombreColum, new Integer(0));
        }else{
          Integer num = (Integer) dups.get(nombreColum);
          int intValor = num.intValue();
          num = new Integer(intValor + 1);
          dups.put(nombreColum, num);
          nombreColum = nombreColum + num;
        }
        nombresColum[col - 1] = nombreColum;
      }
      for (int i = 0;i < resultSets.size();i++){
        while (((ResultSet) resultSets.get(i)).next() ){
          ContenedorDatos cd = new ContenedorDatos(contColum);
          for (int col = 1;col <= contColum;col++){
            if (rellenarNombresColumnas){
              Object value = ((ResultSet) resultSets.get(i)).getObject(col);
              cd.setNombreCampoYValor(col, nombresColum[col - 1], ((ResultSet) resultSets.get(i)).getObject(col));
            }else{
              cd.setObjeto(col, ((ResultSet) resultSets.get(i)).getObject(col));
            }
          }
          listDatos.add(cd);
        }
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
          ((ResultSet) resultSets.get(i)).clearWarnings();
        }
      }
      return listDatos;
    }

    public List getResultadosComoListaContenedorDatos(String sql, Collection colParms, boolean rellenarNombresColumnas) throws SQLException{
      getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
      ResultSetMetaData rsmd = ((ResultSet) this.resultSets.get(0)).getMetaData();
      int contColum = rsmd.getColumnCount();
      List listDatos = new ArrayList();
      String[] nombresColum = new String[contColum];
      Map dups = new HashMap();
      // cambiar el nombre a mapDatos si hay nombres duplicados
      for (int col = 1;col <= contColum;col++){
        String nombreColum = rsmd.getColumnName(col);
        if (!dups.containsKey(nombreColum)){
          dups.put(nombreColum, new Integer(0));
        }else{
          Integer num = (Integer) dups.get(nombreColum);
          int intValor = num.intValue();
          num = new Integer(intValor + 1);
          dups.put(nombreColum, num);
          nombreColum = nombreColum + num;
        }
        nombresColum[col - 1] = nombreColum;
      }
      for (int i = 0;i < resultSets.size();i++){
        while (((ResultSet) resultSets.get(i)).next() ){
          ContenedorDatos cd = new ContenedorDatos(contColum);
          for (int col = 1;col <= contColum;col++){
            if (rellenarNombresColumnas){
              Object value = ((ResultSet) resultSets.get(i)).getObject(col);
              cd.setNombreCampoYValor(col, nombresColum[col - 1], ((ResultSet) resultSets.get(i)).getObject(col));
            }else{
              cd.setObjeto(col, ((ResultSet) resultSets.get(i)).getObject(col));
            }
          }
          listDatos.add(cd);
        }
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
          ((ResultSet) resultSets.get(i)).clearWarnings();
        }
      }
      return listDatos;
    }

    /*************************************************************************/

    /**  @return Vector de Vectores */
    public Vector getResultadosComoVector(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion) throws
SQLException{
      getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
```

```
    int contColum = ((ResultSet) this.resultSets.get(0)).getMetaData().getColumnCount();
    Vector vectorDatos = new Vector();
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        Vector v = new Vector(contColum);
        for (int col = 1;col <= contColum;col++){
          Object valor = ((ResultSet) resultSets.get(i)).getObject(col);
          v.add(valor);
        }
        vectorDatos.add(v);
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    return vectorDatos;
  }

  /**  @return Vector de Vectores */
  public Vector getResultadosComoVector(String sql, Collection colParms) throws SQLException{
    getResultSets(sql, colParms);
    int contColum = ((ResultSet) this.resultSets.get(0)).getMetaData().getColumnCount();
    Vector vectorDatos = new Vector();
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        Vector v = new Vector(contColum);
        for (int col = 1;col <= contColum;col++){
          Object valor = ((ResultSet) resultSets.get(i)).getObject(col);
          v.add(valor);
        }
        vectorDatos.add(v);
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    return vectorDatos;
  }

  public Vector getResultadoComoVector(String sql) throws SQLException{
    getResultSet(sql);
    int colCount = this.resultSet.getMetaData().getColumnCount();
    Vector vectorDatos = new Vector();
    while(this.resultSet.next()){
      Vector v = new Vector(colCount);
      for (int col = 1;col <= colCount;col++){
        v.add(this.resultSet.getObject(col));
      }
      vectorDatos.add(v);
    }
    if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(this.resultSet.getWarnings());
      this.resultSet.clearWarnings();
    }
    return vectorDatos;
  }

  /**  @return Vector de Vectores */
  public Vector getResultadoComoVector(String sql, Collection colParms) throws SQLException{
    getResultSet(sql, colParms);
    int colCount = this.resultSet.getMetaData().getColumnCount();
    Vector vectorDatos = new Vector();
    while(this.resultSet.next()){
      Vector v = new Vector(colCount);
      for (int col = 1;col <= colCount;col++){
        v.add(this.resultSet.getObject(col));
      }
      vectorDatos.add(v);
    }
    if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(this.resultSet.getWarnings());
      this.resultSet.clearWarnings();
    }
    return vectorDatos;
  }

  /*********************************************************************/
```

```
  /**  @return Map of Lists. Cada clave es un nombre de listDatos y cada valor es un List de valores de listDatos */
  public Map getResultadosComoMap(String sql, Collection colParms) throws SQLException{
    getResultSets(sql, colParms);
    ResultSetMetaData rsmd = ((ResultSet) this.resultSets.get(0)).getMetaData();
    int colCount = rsmd.getColumnCount();
    Map mapDatos = new LinkedHashMap(colCount);
    List[] listDatos = new ArrayList[colCount];
    for (int i = 0;i < listDatos.length;i++){
      listDatos[i] = new ArrayList();
    }
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        for (int col = 1;col <= colCount;col++){
          Object valor = ((ResultSet) resultSets.get(i)).getObject(col);
          listDatos[col - 1].add(valor);
        }
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    Map dups = new HashMap();
    for (int col = 1;col <= colCount;col++){
      String nombreColum = rsmd.getColumnName(col);
      if (!dups.containsKey(nombreColum)){
        dups.put(nombreColum, new Integer(0));
      }else{
        Integer i = (Integer) dups.get(nombreColum);
        int intValor = i.intValue();
        i = new Integer(intValor + 1);
        dups.put(nombreColum, i);
        nombreColum = nombreColum + i;
      }
      mapDatos.put(nombreColum, listDatos[col - 1]);
    }
    return mapDatos;
  }

  /**  @return Map of Lists. Cada clave es un nombre de listDatos y cada valor es una List de valores de listDatos */
  public Map getResultadosComoMap(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion) throws
SQLException{
    getResultSets(sql, colParms, numeroMaximoParametros, marcadorPosicion);
    ResultSetMetaData rsmd = ((ResultSet) this.resultSets.get(0)).getMetaData();
    int colCount = rsmd.getColumnCount();
    Map mapDatos = new LinkedHashMap(colCount);
    List[] listDatos = new ArrayList[colCount];
    for (int i = 0;i < listDatos.length;i++){
      listDatos[i] = new ArrayList();
    }
    for (int i = 0;i < resultSets.size();i++){
      while (((ResultSet) resultSets.get(i)).next() ){
        for (int col = 1;col <= colCount;col++){
          Object valor = ((ResultSet) resultSets.get(i)).getObject(col);
          listDatos[col - 1].add(valor);
        }
      }
      if (isAdvertenciasProcesadas()){
        rellenarAdvertencias(((ResultSet) resultSets.get(i)).getWarnings());
        ((ResultSet) resultSets.get(i)).clearWarnings();
      }
    }
    Map dups = new HashMap();
    for (int col = 1;col <= colCount;col++){
      String nombreColum = rsmd.getColumnName(col);
      if (!dups.containsKey(nombreColum)){
        dups.put(nombreColum, new Integer(0));
      }else{
        Integer i = (Integer) dups.get(nombreColum);
        int intValor = i.intValue();
        i = new Integer(intValor + 1);
        dups.put(nombreColum, i);
        nombreColum = nombreColum + i;
      }
      mapDatos.put(nombreColum, listDatos[col - 1]);
    }
    return mapDatos;
  }

  /**  @return Map of Lists. Cada clave es un nombre de listDatos y cada valor es una List de valores de listDatos */
```

```
public Map getResultadoComoMap(String sql, Collection colParms) throws SQLException{
   getResultSet(sql, colParms);
   ResultSetMetaData rsmd = this.resultSet.getMetaData();
   int colCount = rsmd.getColumnCount();
   Map mapDatos = new LinkedHashMap(colCount);
   List[] listDatos = new ArrayList[colCount];
   for (int i = 0;i < listDatos.length;i++){
      listDatos[i] = new ArrayList();
   }
   while(this.resultSet.next()){
      for (int col = 1;col <= colCount;col++){
         Object valor = this.resultSet.getObject(col);
         listDatos[col - 1].add(valor);
      }
   }
   Map dups = new HashMap();
   for (int col = 1;col <= colCount;col++){
      String nombreColum = rsmd.getColumnName(col);
      if (!dups.containsKey(nombreColum)){
         dups.put(nombreColum, new Integer(0));
      }else{
         Integer i = (Integer) dups.get(nombreColum);
         int intValor = i.intValue();
         i = new Integer(intValor + 1);
         dups.put(nombreColum, i);
         nombreColum = nombreColum + i;
      }
      mapDatos.put(nombreColum, listDatos[col - 1]);
   }
   if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(this.resultSet.getWarnings());
      this.resultSet.clearWarnings();
   }
   return mapDatos;
}

/**  @return Map of Lists. Cada clave es un nombre de listDatos y cada valor es una List de valores de listDatos */
public Map getResultadoComoMap(String sql) throws SQLException{
   getResultSet(sql);
   ResultSetMetaData rsmd = this.resultSet.getMetaData();
   int colCount = rsmd.getColumnCount();
   Map mapDatos = new LinkedHashMap(colCount);
   List[] listDatos = new ArrayList[colCount];
   for (int i = 0;i < listDatos.length;i++){
      listDatos[i] = new ArrayList();
   }
   while(this.resultSet.next()){
      for (int col = 1;col <= colCount;col++){
         Object valor = this.resultSet.getObject(col);
         listDatos[col - 1].add(valor);
      }
   }
   Map dups = new HashMap();
   for (int col = 1;col <= colCount;col++){
      String nombreColum = rsmd.getColumnName(col);
      if (!dups.containsKey(nombreColum)){
         dups.put(nombreColum, new Integer(0));
      }else{
         Integer i = (Integer) dups.get(nombreColum);
         int intValor = i.intValue();
         i = new Integer(intValor + 1);
         dups.put(nombreColum, i);
         nombreColum = nombreColum + i;
      }
      mapDatos.put(nombreColum, listDatos[col - 1]);
   }
   if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(this.resultSet.getWarnings());
      this.resultSet.clearWarnings();
   }
   return mapDatos;
}

/*********************************************************************/

/** Leer la propiedad de resultSet.
 * @return Value of property resultSet.
 *
 */
```

```
    public java.sql.ResultSet getResultSet(String sql) throws SQLException{
        if (resultSet != null && isCerrarResultSet()){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSet()::closing
ResultSet: " + resultSet + " \n");
            resultSet.close();
        }
        resultSet = this.getStatement(sql).executeQuery(sql);
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSet()::sql: \n" + sql +
" \n");
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(resultSet.getWarnings());
            this.resultSet.clearWarnings();
        }
        return resultSet;
    }

    public java.sql.ResultSet getResultSet(String sql, Collection colParms) throws SQLException{
        if (resultSet != null && isCerrarResultSet()){
            resultSet.close();
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSet()::cerrando
ResultSet: " + resultSet + " \n");
        }
        resultSet = this.getPreparedStatement(sql, colParms).executeQuery();
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSet(): " + resultSet + "
\n");
        if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(resultSet.getWarnings());
        }
        return resultSet;
    }

    public java.sql.ResultSet[] getResultSets(String sql, Collection colParms, int numeroMaximoParametros, String marcadorPosicion) throws
SQLException{
        this.numeroMaximoParametros = numeroMaximoParametros;
        this.marcadorPosicion = marcadorPosicion;
        return getResultSets(sql, colParms);
    }

    public java.sql.ResultSet[] getResultSets(String sql, Collection colParms) throws SQLException{
        /**
         *  Este es el método más conveniente para construir sentencias con la cláusula SQL 'IN'
         *  si la cantidad de argumentos que se usarán en la sentencia con la cláusula "IN" es
         *  más grande que el número de máximo de parámetros que la base de datos puede tratar.
         *  Generará varios objetos ResultSet.
         *  La lista de argumentos que es más grande que el número de máximo de los parámetros
         *  que la base de datos puede tratar debe estar en el último lugar en la variable colParms.
         *  También, tanto marcadorPosicion como numeroMaximoParametros deben ser el asignados.
         *
         *  Ejemplo de uso:
         *   String sql = "SELECT * FROM CUSTOMER WHERE LAST_NAME <> ? AND CUSTOMER_ID IN (:#)";
         *   Collection argsList = new ArrayList();
         *   List custId = new ArrayList();
         *   custId.add(new Integer(0));
         *   custId.add(new Integer(1));
         *   custId.add(new Integer(2));
         *   custId.add(new Integer(3));
         *   custId.add(new Integer(4));
         *   custId.add(new Integer(5));
         *   custId.add(new Integer(6));
         *   custId.add(new Integer(7));
         *   argsList.add("Ricardo García");
         *   argsList.add(custId);
         *   factory.getDBServices().setNumeroMaximoParametros(3);
         *   factory.getDBServices().setMarcadorPosicion(":#");
         *   factory.load(factory.getDBServices().getResultSets(sql, argsList));
         *   factory.getDBServices().release(true);
         */
        closeResults();
        resultSets = new ArrayList();
        Collection nuevaListaArgs = new ArrayList();
        String nuevaSQL = null;
        Collection nuevaListaArgs1 = null;
        boolean cargado = false;
        boolean cerrarResultOriginal = this.cerrarResultSet;
        this.cerrarResultSet = false;
        int parametrosCompuestos = 0;
        int longitudParams = colParms.size();
        Iterator testParamsIter = colParms.iterator();
        while (testParamsIter.hasNext()){
            Object obj = testParamsIter.next();
```

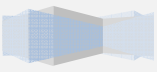
```
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSets()::obj:" + obj +
"::instanceof Collection: " + (obj instanceof Collection) + " \n");
        if (obj instanceof Collection){  // es un parametro compuesto
           parametrosCompuestos++;
        }
     }
     String[] marcadoresPosicionTest = sql.split(this.marcadorPosicion);
     if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSets()::sql: \n" + sql +
"::marcadoresPosicionTest: " + Arrays.asList(marcadoresPosicionTest) + " \n");
     if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSets()::sql: \n" + sql +
"::parametrosCompuestos: " + parametrosCompuestos + "::marcadoresPosicionTest.length: " + marcadoresPosicionTest.length  + " \n");
     if ((marcadoresPosicionTest.length - 1) != parametrosCompuestos){
        throw new SQLException("Number of placeholders in sql is not equal to number of compound parameters");
     }
     try{
        Iterator iter1 = colParms.iterator();
        int contParams = 0;
        while (iter1.hasNext()){
           Object argActual = iter1.next();
           if (argActual instanceof Collection){  // es un parametro compuesto
              if (numeroMaximoParametros < 0){
                 throw new SQLException("Invalid maxNumberOfParams: " + numeroMaximoParametros);
              }
              List lista = new ArrayList((Collection) argActual);
              int thisParmSize = lista.size();
              if (thisParmSize > numeroMaximoParametros){
                 if (contParams < (longitudParams - 1)){
                    throw new SQLException("Only the last entry can have more arguments then Max Number of Parameters");
                 }
                 nuevaSQL = sql;
                 StringBuffer sb = new StringBuffer();
                 Iterator iter3 = lista.iterator();
                 int cont = 0;
                 int total = 0;
                 List temp = new ArrayList();
                 Object paramValor = null;
                 for (int i = 0;i < thisParmSize;i++){
                    if (cont < (numeroMaximoParametros)){
                       paramValor = lista.get(i);
                       sb.append("?,");
                       temp.add(paramValor);
                       if (i == (thisParmSize - 1)){
                          int idx = sb.length() - 1;
                          char c = sb.charAt(idx);
                          if (c == ','){
                             sb.deleteCharAt(idx);
                          }
                          nuevaSQL = sql.replaceFirst(marcadorPosicion, sb.toString());
                          nuevaListaArgs1 = new ArrayList(nuevaListaArgs);
                          nuevaListaArgs1.addAll(temp);
                          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::getResultSets()::sql: " + nuevaSQL + "::args: " + nuevaListaArgs1 + " \n");
                          resultSets.add(this.getPreparedStatement(nuevaSQL, nuevaListaArgs1).executeQuery());
                          cargado = true;
                          break;
                       }
                       cont++;
                    }else if (cont == (numeroMaximoParametros)){
                       int idx = sb.length() - 1;
                       char c = sb.charAt(idx);
                       if (c == ','){
                          sb.deleteCharAt(idx);
                       }
                       nuevaSQL = sql.replaceFirst(marcadorPosicion, sb.toString());
                       nuevaListaArgs1 = new ArrayList(nuevaListaArgs);
                       nuevaListaArgs1.addAll(temp);
                       if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::getResultSets()::sql: " + nuevaSQL + "::args: " + nuevaListaArgs1 + " \n");
                       resultSets.add(this.getPreparedStatement(nuevaSQL, nuevaListaArgs1).executeQuery());
                       cargado = true;
                       temp.clear();
                       sb.setLength(0);
                       nuevaSQL = sql;
                       cont = 0;
                       i--;
                    }
                 }
              }else{
                 StringBuffer sb = new StringBuffer();
```

```
                Iterator iter2 = lista.iterator();
                while (iter2.hasNext()){
                  sb.append("?,");
                  nuevaListaArgs.add(iter2.next());
                }
                sb.deleteCharAt(sb.lastIndexOf(","));
                sql = sql.replaceFirst(marcadorPosicion, sb.toString());
              }
            }else{ // es un parametro corriente
              nuevaListaArgs.add(argActual);
            }
            contParams++;
          }
          if (!cargado){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::getResultSets()::sql: \n" +
sql + "::args: " + nuevaListaArgs + " \n");
            resultSets.add(this.getPreparedStatement(sql, nuevaListaArgs).executeQuery());
          }
        }catch(Exception e){
          e.printStackTrace();
          throw new SQLException(e.getMessage());
        }
        setCerrarResultSet(cerrarResultOriginal);
        return (java.sql.ResultSet[]) resultSets.toArray(new ResultSet[0]);
      }

      public List ejecutarProcedimientoAlmacenado(String sql, Parametros params, int proccessResult) throws SQLException{
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarProcedimientoAlmacenado()::sql: \n" + sql + "\n");
        List valorDevuelto = new ArrayList();
        try {
          callStatement = getConnection().prepareCall(sql);
          for (int i = 1;i <= params.getParameterCount();i++){
            int paramMode = params.getParameterMode(i);
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarProcedimientoAlmacenado()::paramMode: " + paramMode + "::paramIndex: " + i + "::paramTipo: " + params.getParameterType(i) +
"::parameterName: " + params.getParameterName(i) + "::valorParms: " + params.getParameter(i) + "\n");
            if (paramMode == Parametros.parameterModeIn){
              callStatement.setObject(i, params.getParameter(i));
            }else if (paramMode == Parametros.parameterModeOut){
              callStatement.registerOutParameter(i, params.getParameterType(i));
            }else if (paramMode == Parametros.parameterModeInOut){
              callStatement.registerOutParameter(i, params.getParameterType(i));
              callStatement.setObject(i, params.getParameter(i));
            }
          }

          ResultSet result = null;
          int filasAfectadas = 0;
          boolean hasResult = callStatement.execute();
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarProcedimientoAlmacenado()::hasResult: " + hasResult + "\n");

          // procesar ResultSet(s) con/sin updateCount
          while (hasResult || filasAfectadas != -1){
            if (hasResult == true){
              result = callStatement.getResultSet();
              try{
                if (proccessResult == IGNORAR_RESULTADO){
                  // no hacer nada
                }else if (proccessResult == RESULTADO_LISTA_CONTENEDOR_DATOS){
                  ResultSetMetaData rsmd = result.getMetaData();
                  int contColum = rsmd.getColumnCount();
                  List listDatos = new ArrayList();
                  String[] nombresColum = new String[contColum];
                  Map dups = new HashMap();
                  // cambiar el nombre de mapDatos si hay nombres duplicados
                  for (int col = 1;col <= contColum;col++){
                    String nombreColum = rsmd.getColumnName(col);
                    if (!dups.containsKey(nombreColum)){
                      dups.put(nombreColum, new Integer(0));
                    }else{
                      Integer num = (Integer) dups.get(nombreColum);
                      int intValor = num.intValue();
                      num = new Integer(intValor + 1);
                      dups.put(nombreColum, num);
                      nombreColum = nombreColum + num;
                    }
                    nombresColum[col - 1] = nombreColum;
                  }
                  while(result.next()){
```

```
                    ContenedorDatos cd = new ContenedorDatos(contColum);
                    for (int col = 1;col <= contColum;col++){
                      cd.setNombreCampoYValor(col, nombresColum[col - 1], result.getObject(col));
                    }
                    listDatos.add(cd);
                  }
                  if (isAdvertenciasProcesadas()){
                    rellenarAdvertencias(result.getWarnings());
                    result.clearWarnings();
                  }
                  valorDevuelto.add(listDatos);
                }else if (proccessResult == RESULTADO_VECTOR){
                  int contColum = result.getMetaData().getColumnCount();
                  Vector vectorDatos = new Vector();
                  while(result.next()){
                    Vector v = new Vector(contColum);
                    for (int col = 1;col <= contColum;col++){
                      v.add(result.getObject(col));
                    }
                    vectorDatos.add(v);
                  }
                  if (isAdvertenciasProcesadas()){
                    rellenarAdvertencias(result.getWarnings());
                    result.clearWarnings();
                  }
                  valorDevuelto.add(vectorDatos);
                }else if (proccessResult == RESULTADO_MAP){
                  ResultSetMetaData rsmd = result.getMetaData();
                  int contColum = rsmd.getColumnCount();
                  Map mapDatos = new LinkedHashMap(contColum);
                  List[] listDatos = new ArrayList[contColum];
                  for (int i = 0;i < listDatos.length;i++){
                    listDatos[i] = new ArrayList();
                  }
                  while(result.next()){
                    for (int col = 1;col <= contColum;col++){
                      Object obj = result.getObject(col);
                      listDatos[col - 1].add(obj);
                    }
                  }
                  Map dups = new HashMap();
                  for (int col = 1;col <= contColum;col++){
                    String nombreColum = rsmd.getColumnName(col);
                    if (!dups.containsKey(nombreColum)){
                      dups.put(nombreColum, new Integer(0));
                    }else{
                      Integer i = (Integer) dups.get(nombreColum);
                      int intValor = i.intValue();
                      i = new Integer(intValor + 1);
                      dups.put(nombreColum, i);
                      nombreColum = nombreColum + i;
                    }
                    mapDatos.put(nombreColum, listDatos[col - 1]);
                  }
                  if (isAdvertenciasProcesadas()){
                    rellenarAdvertencias(result.getWarnings());
                    result.clearWarnings();
                  }
                  valorDevuelto.add(mapDatos);

                }else if (proccessResult == RESULTADO_ROWSET){
                  javax.sql.rowset.CachedRowSet crs = (javax.sql.rowset.CachedRowSet) Class.forName(cachedRowSetClassName).newInstance();
                  crs.populate(result);
                  valorDevuelto.add(crs);
                }
              }finally{
                if (isAdvertenciasProcesadas()){
                  rellenarAdvertencias(result.getWarnings());
                  result.clearWarnings();
                }
                if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarProcedimientoAlmacenado()::cerrando ResultSet: " + result + " \n");
                result.close();
                result = null;
              }
            }else{
              filasAfectadas = callStatement.getUpdateCount();
              if (filasAfectadas != -1){
                valorDevuelto.add(new Integer(filasAfectadas));
```

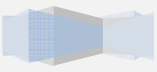
```
            }
          }
          hasResult = callStatement.getMoreResults();
        }

        // procesar parametros tipo: OUT y/o INOUT
        for (int i = 1;i <= params.getParameterCount();i++){
          int paramMode = params.getParameterMode(i);
          if (paramMode == Parametros.parameterModeOut || paramMode == Parametros.parameterModeInOut){
            Object obj = callStatement.getObject(i);
            params.setParameter(i, obj);
          }
        }
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(callStatement.getWarnings());
          this.callStatement.clearWarnings();
        }
        valorDevuelto.add(params);
      } catch (Exception e){
        e.printStackTrace();
        throw new SQLException(e.getMessage());
      }finally{
        if (isCerrarStatementTrasEjecucion()){
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarProcedimientoAlmacenado()::cerrando CallableStatement: " + callStatement + " \n");
          if (callStatement != null){
            callStatement.close();
            callStatement = null;
          }
        }
      }
      return valorDevuelto;
    }

    public int[] ejecutarBatch(String sql, Collection batch) throws SQLException{
      int[] rowsAffected;
      try{
        this.prepararStatement(sql);
        Iterator batchIter = batch.iterator();
        while(batchIter.hasNext()){
          Collection params = (Collection) batchIter.next();
          Iterator paramsIter = params.iterator();
          int parmIdx = 1;
          while(paramsIter.hasNext()){
            Object param = paramsIter.next();
            if (param != null){
              if (param.getClass().getName().equals("Date")){
                param = new java.sql.Date(((java.util.Date) param).getTime());
              }
              preparedStatement.setObject(parmIdx, param);
            }else{
              preparedStatement.setNull(parmIdx, Types.JAVA_OBJECT);
            }
            parmIdx++;
          }
          preparedStatement.addBatch();
        }
        rowsAffected = this.preparedStatement.executeBatch();
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::executeBatch(" + sql +
")::batch: " + batch + "::filasAfectadas: " + Arrays.asList(rowsAffected) + "\n");
      }finally{
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(preparedStatement.getWarnings());
          this.preparedStatement.clearWarnings();
        }
        if (isCerrarStatementTrasEjecucion()){
          if (preparedStatement != null){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::executeBatch()::cerrando PreparedStatement: " + preparedStatement + " \n");
            preparedStatement.close();
            preparedStatement = null;
          }
        }
      }
      return rowsAffected;
    }

    public int[] ejecutarBatch(Collection batch) throws SQLException{
      int[] filasAfectadas;
      createStatement();
      try{
```

```
        Iterator batchIter = batch.iterator();
        while(batchIter.hasNext()){
          String sql = (String) batchIter.next();
          this.statement.addBatch(sql);
        }
        filasAfectadas = this.statement.executeBatch();
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::ejecutarBatch()::batch: " +
batch + "::filasAfectadas: " + Arrays.asList(filasAfectadas) +"\n");
      }finally{
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(statement.getWarnings());
          this.statement.clearWarnings();
        }
        if (isCerrarStatementTrasEjecucion()){
          if (statement != null){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarBatch()::cerrando Statement: " + statement + " \n");
              statement.close();
              statement = null;
          }
        }
      }
      return filasAfectadas;
    }

    public int ejecutar(String sql) throws SQLException{
      int filasAfectadas = -1;
      try{
        filasAfectadas = this.getStatement(sql).executeUpdate(sql);
      }finally{
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(statement.getWarnings());
          this.statement.clearWarnings();
        }
        if (isCerrarStatementTrasEjecucion()){
          if (statement != null){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarBatch()::cerrando Statement: " + statement + " \n");
              statement.close();
              statement = null;
          }
        }
      }
      this.setFilasAfectadasPorTransaccion(filasAfectadasPorTransaccion + filasAfectadas);
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::ejecutar()::sql: \n" + sql +
"::filasAfectadas: " + filasAfectadas + "\n");
      return filasAfectadas;
    }

    public int ejecutar(String sql, Collection colParms) throws SQLException{
      int filasAfectadas = -1;
      try{
        filasAfectadas = this.getPreparedStatement(sql, colParms).executeUpdate();
      }finally{
        if (isAdvertenciasProcesadas()){
          rellenarAdvertencias(statement.getWarnings());
          this.statement.clearWarnings();
        }
        if (isCerrarStatementTrasEjecucion()){
          if (preparedStatement != null){
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::ejecutarBatch()::cerrando PreparedStatement: " + preparedStatement + " \n");
              preparedStatement.close();
              preparedStatement = null;
          }
        }
      }
      this.setFilasAfectadasPorTransaccion(filasAfectadasPorTransaccion + filasAfectadas);
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::ejecutar()::sql: \n" + sql + "
params: " + colParms + "::rowsAffected: " + filasAfectadas + "\n");
      return filasAfectadas;
    }

    /** Asignar nueva propiedad a resultSet.
     * @param resultSet New valor of property resultSet.
     *
     */
    public void setResultSet(java.sql.ResultSet resultSet) {
      this.resultSet = resultSet;
```

```
  }

  protected void createStatement() throws SQLException{
    statement = this.getConnection().createStatement(getResultadoSetType(), getConcurrencia());
    statement.setEscapeProcessing(this.getEscapeProcessing());
    statement.setFetchSize(this.getFetchLongitud());
    statement.setMaxRows(this.getMaxFilas());
    statement.setMaxFieldSize(this.getMaxLongitudCampo());
    statement.setQueryTimeout(this.getQueryTiempoLimite());
    if (this.getNombreCursor() != null){
      statement.setCursorName(this.getNombreCursor());
    }
  }

  /** Leer la propiedad de statement.
   * @return Value of property statement.
   *
   */
  public java.sql.Statement getStatement(String sql) throws SQLException{
    if (sql == null || sql.trim().length() == 0) {
      throw new SQLException("SQL String is empty or null");
    }
    int idx = -1;
    if (almacenStatements){  // Statement to be reused, otherwise must be closed and set to null
      idx = sentenciasId.indexOf(sql);
      if (idx >= 0){
        statement = (Statement) sentenciasAlmacenamiento.get(idx);
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
getStatement() ============ encontrado almacen. statement: \n" + sql + "::" + statement + " \n");
        if (statement == null){
          sentenciasId.remove(idx);
          sentenciasAlmacenamiento.remove(idx); //??
          idx = -1;
        }
      }
    }
    if (idx == -1){
      createStatement();
    }
    if (almacenStatements){
      if (idx == -1){
        idx = addAlmacenStatement(sql, statement);
        contadorSentenciasAlmacenadas.add(new Integer(1));
      }else{
        // ¿Cuán a menudo fue llamada esta instrucción?
        Object cont = contadorSentenciasAlmacenadas.get(idx);
        if (cont == null){
          contadorSentenciasAlmacenadas.add(new Integer(1));
        }else{
          contadorSentenciasAlmacenadas.set(idx, new Integer(((Integer) cont).intValue() + 1));
        }
      }
    }
    if (isAdvertenciasProcesadas()){
      rellenarAdvertencias(statement.getWarnings());
      statement.clearWarnings();
    }
    return statement;
  }

  protected void closeResults() throws SQLException{
    if (resultSets != null){
      for (int i = 0;i < resultSets.size();i++){
        Object obj = resultSets.get(i);
        if (obj != null){
          ((ResultSet) obj).close();
          obj = null;
        }
      }
      resultSets = null;
    }
  }

  /** Asignar nueva propiedad a statement.
   * @param statement New valor of property statement.
   *
   */
  public void setStatement(java.sql.Statement statement) {
    this.statement = statement;
  }
```

```
   public void release(boolean conexionCerrada) throws SQLException{
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================ RELEASE
============ conexionCerrada: " + conexionCerrada + " \n");
      if (resultSet != null){
         resultSet.close();
         resultSet = null;
      }
      closeResults();
      if (!almacenStatements){
         if (statement != null){
            statement.close();
            statement = null;
         }
         if (preparedStatement != null){
            preparedStatement.close();
            preparedStatement = null;
         }
         if (callStatement != null){
            callStatement.close();
            callStatement = null;
         }
      }
      if (advertencias != null){
         advertencias.clear();
      }
      if (connection != null){
         if (conexionCerrada == true){
            borrarAlmacenStatement();
            if (proccessTransactionOnCloseConnection == ROLLBACK){
               connection.rollback();
            }else if (proccessTransactionOnCloseConnection == COMMIT){
               connection.commit();
            }
            connection.close();
            setConexionCerrada(true);
            connection = null;
            this.setTransaccionEnCurso(false);
            this.setConexionId("<none>");
         }else{
            //no hacer nada
         }
      }
   }

   public void beginTran() throws SQLException{
      if (EJBContext == null){
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================ BEGIN
TRAN ============ anterior connectionId: " + conexionId + " \n");
         this.setFilasAfectadasPorTransaccion(0);
         this.getConnection().setAutoCommit(false);
         this.setConexionId(this.connection.toString());
         this.setTransaccionEnCurso(true);
         if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.connection.getWarnings());
            this.connection.clearWarnings();
         }
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================ BEGIN
TRAN ============ actual connectionId: " + conexionId + " \n");
      }else{
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
EJBContext: " + EJBContext + " \n");
      }
   }

   public void commitTran() throws SQLException{
      if (EJBContext == null){
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================ COMMIT
TRAN ============= actual connectionId: " + conexionId + " \n");
         this.getConnection().commit();
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
COMMITTED ================\n");
         this.getConnection().setAutoCommit(true);//?? Sybase bug
         this.setTransaccionEnCurso(false);
         this.setConexionId("<none>");
         if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.connection.getWarnings());
            this.connection.clearWarnings();
         }
      }else{
```

```
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
EJBContext: " + EJBContext + " \n");
      }
   }

   public void rollbackTran() throws SQLException{
      if (EJBContext != null){
         try{
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
EJBContext: " + EJBContext +".setRollbackOnly ============= connectionId: " + conexionId + " \n");
            Method m = EJBContext.getClass().getMethod("setRollbackOnly", new Class[]{});
            m.invoke(EJBContext, new Object[]{});
         }catch (InvocationTargetException ete) {
            throw new SQLException(ete.getTargetException().getMessage());
         }catch(Exception e){
            throw new SQLException(e.getMessage());
         }
      }else{
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
ROLLBACK TRAN ============= connectionId: " + conexionId + " \n");
         this.getConnection().rollback();
         if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================ AFTER
ROLLBACK =========== connectionId: " + conexionId + " \n");
         this.getConnection().setAutoCommit(true);
         this.setTransaccionEnCurso(false);
         this.setConexionId("<none>");
         if (isAdvertenciasProcesadas()){
            rellenarAdvertencias(this.connection.getWarnings());
            this.connection.clearWarnings();
         }
      }
   }

   /** Leer la propiedad de nombreOrigenDatos.
    * @return Value of property nombreOrigenDatos.
    *
    */
   public String getNombreOrigenDatos() {
      return nombreOrigenDatos;
   }

   /** Asignar nueva propiedad a nombreOrigenDatos.
    * @param nombreOrigenDatos New valor of property nombreOrigenDatos.
    *
    */
   public void setNombreOrigenDatos(String nombreOrigenDatos) {
      this.nombreOrigenDatos = nombreOrigenDatos;
   }

   /** Leer la propiedad de fetchLongitud.
    * @return Value of property fetchLongitud.
    *
    */
   public int getFetchLongitud() {
      return fetchLongitud;
   }

   /** Asignar nueva propiedad a fetchLongitud.
    * @param fetchLongitud New valor of property fetchLongitud.
    *
    */
   public void setFetchLongitud(int fetchLongitud) {
      this.fetchLongitud = fetchLongitud;
   }

   /** Leer la propiedad de fetchDireccion.
    * @return Value of property fetchDireccion.
    *
    */
   public int getFetchDireccion() {
      return fetchDireccion;
   }

   /** Asignar nueva propiedad a fetchDireccion.
    * @param fetchDireccion New valor of property fetchDireccion.
    *
    */
   public void setFetchDireccion(int fetchDireccion) {
      this.fetchDireccion = fetchDireccion;
   }
```

```
/** Leer la propiedad de queryTiempoLimite.
 * @return Value of property queryTiempoLimite.
 *
 */
public int getQueryTiempoLimite() {
   return queryTiempoLimite;
}

/** Asignar nueva propiedad a queryTiempoLimite.
 * @param queryTiempoLimite New valor of property queryTiempoLimite.
 *
 */
public void setQueryTiempoLimite(int queryTiempoLimite) {
   this.queryTiempoLimite = queryTiempoLimite;
}

/** Leer la propiedad de maxFilas.
 * @return Value of property maxFilas.
 *
 */
public int getMaxFilas() {
   return maxFilas;
}

/** Asignar nueva propiedad a maxFilas.
 * @param maxFilas New valor of property maxFilas.
 *
 */
public void setMaxFilas(int maxFilas) {
   this.maxFilas = maxFilas;
}

/** Leer la propiedad de maxLongitudCampo.
 * @return Value of property maxLongitudCampo.
 *
 */
public int getMaxLongitudCampo() {
   return maxLongitudCampo;
}

/** Asignar nueva propiedad a maxLongitudCampo.
 * @param maxLongitudCampo New valor of property maxLongitudCampo.
 *
 */
public void setMaxLongitudCampo(int maxLongitudCampo) {
   this.maxLongitudCampo = maxLongitudCampo;
}

/** Leer la propiedad de escapeProcessing.
 * @return Value of property escapeProcessing.
 *
 */
public boolean getEscapeProcessing() {
   return escapeProcessing;
}

/** Asignar nueva propiedad a escapeProcessing.
 * @param escapeProcessing New valor of property escapeProcessing.
 *
 */
public void setEscapeProcessing(boolean escapeProcessing) {
   this.escapeProcessing = escapeProcessing;
}

/** Leer la propiedad de resultadoSetType.
 * @return Value of property resultadoSetType.
 *
 */
public int getResultadoSetType() {
   return resultadoSetType;
}

/** Asignar nueva propiedad a resultadoSetType.
 * @param resultadoSetType New valor of property resultadoSetType.
 *
 */
public void setResultadoSetType(int resultadoSetType) {
   this.resultadoSetType = resultadoSetType;
}
```

```
/** Leer la propiedad de concurrencia.
 * @return Value of property concurrencia.
 *
 */
public int getConcurrencia() {
   return concurrencia;
}

/** Asignar nueva propiedad a concurrencia.
 * @param concurrencia New valor of property concurrencia.
 *
 */
public void setConcurrencia(int concurrencia) {
   this.concurrencia = concurrencia;
}

/** Leer la propiedad de conexionCerrada.
 * @return Value of property conexionCerrada.
 *
 */
public boolean isConexionCerrada() {
   return conexionCerrada;
}

/** Asignar nueva propiedad a conexionCerrada.
 * @param conexionCerrada New valor of property conexionCerrada.
 *
 */
protected void setConexionCerrada(boolean conexionCerrada) {
   this.conexionCerrada = conexionCerrada;
}

/** Leer la propiedad de maxFilasProcesarTransaccion.
 * @return Value of property maxFilasProcesarTransaccion.
 *
 */
public int getMaxFilasProcesarTransaccion() {
   return maxFilasProcesarTransaccion;
}

/** Asignar nueva propiedad a maxFilasProcesarTransaccion.
 * @param maxFilasProcesarTransaccion New valor of property maxFilasProcesarTransaccion.
 *
 */
public void setMaxFilasProcesarTransaccion(int maxFilasProcesarTransaccion) throws SQLException{
   if (maxFilasProcesarTransaccion < 0){
      throw new SQLException("Invalid maxRowsToProccesInTransaction value");
   }
   this.maxFilasProcesarTransaccion = maxFilasProcesarTransaccion;
}

/** Leer la propiedad de filasAfectadasPorTransaccion.
 * @return Value of property filasAfectadasPorTransaccion.
 *
 */
public int getFilasAfectadasPorTransaccion() {
   return filasAfectadasPorTransaccion;
}

/** Asignar nueva propiedad a filasAfectadasPorTransaccion.
 * @param filasAfectadasPorTransaccion New valor of property filasAfectadasPorTransaccion.
 *
 */
public void setFilasAfectadasPorTransaccion(int filasAfectadasPorTransaccion) throws SQLException{
   this.filasAfectadasPorTransaccion = filasAfectadasPorTransaccion;
   if (this.maxFilasProcesarTransaccion != 0 &&
         this.filasAfectadasPorTransaccion >= this.maxFilasProcesarTransaccion){
      try{
         this.commitTran();
         this.release(false);
         this.beginTran();
      }catch(SQLException e){
         this.rollbackTran();
         throw e;
      }
   }
}

/** Leer la propiedad de conexionProperties.
 * @return Value of property conexionProperties.
```

```
     *
     */
    public Properties getConexionProperties() {
        return conexionProperties;
    }

    /** Asignar nueva propiedad a conexionProperties.
     * @param conexionProperties New valor of property conexionProperties.
     *
     */
    public void setConexionProperties(Properties conexionProperties) {
        this.conexionProperties = conexionProperties;
    }

    /** Leer la propiedad de cerrarStatementTrasEjecucion.
     * @return Value of property cerrarStatementTrasEjecucion.
     *
     */
    public boolean isCerrarStatementTrasEjecucion() {
        return cerrarStatementTrasEjecucion;
    }

    /** Asignar nueva propiedad a cerrarStatementTrasEjecucion.
     * @param cerrarStatementTrasEjecucion New valor of property cerrarStatementTrasEjecucion.
     *
     */
    public void setCerrarStatementTrasEjecucion(boolean cerrarStatementTrasEjecucion) {
        this.cerrarStatementTrasEjecucion = cerrarStatementTrasEjecucion;
    }

    /** Leer la propiedad de conexionId.
     * @return Value of property conexionId.
     *
     */
    public String getConexionId() {
        return conexionId;
    }

    /** Asignar nueva propiedad a conexionId.
     * @param conexionId New valor of property conexionId.
     *
     */
    public void setConexionId(String conexionId) {
        this.conexionId = conexionId;
    }

    /** Leer la propiedad de transaccionEnCurso.
     * @return Value of property transaccionEnCurso.
     *
     */
    public boolean isTransaccionEnCurso() {
        return transaccionEnCurso;
    }

    /** Asignar nueva propiedad a transaccionEnCurso.
     * @param transaccionEnCurso New valor of property transaccionEnCurso.
     *
     */
    public void setTransaccionEnCurso(boolean transaccionEnCurso) {
        this.transaccionEnCurso = transaccionEnCurso;
    }

    /**
     * Leer la propiedad de debug.
     * @return Value of property debug.
     */
    public boolean isDebug() {
        return debug;
    }

    /**
     * Asignar nueva propiedad a debug.
     * @param debug New valor of property debug.
     */
    public void setDebug(boolean debug) {
        this.debug = debug;
    }

    /**
```

```
 * Leer la propiedad de resetOrigenDatos.
 * @return Value of property resetOrigenDatos.
 */
public boolean isResetOrigenDatos() {
   return resetOrigenDatos;
}

/**
 * Asignar nueva propiedad a resetOrigenDatos.
 * @param resetOrigenDatos New valor of property resetOrigenDatos.
 */
public void setResetOrigenDatos(boolean resetOrigenDatos) {
   this.resetOrigenDatos = resetOrigenDatos;
}

public String getMarcadorPosicion() {
   return marcadorPosicion;
}

public void setMarcadorPosicion(String marcadorPosicion) {
   this.marcadorPosicion = marcadorPosicion;
}

public int getNumeroMaximoParametros() {
   return numeroMaximoParametros;
}

public void setNumeroMaximoParametros(int numeroMaximoParametros) {
   this.numeroMaximoParametros = numeroMaximoParametros;
}

public List getAdvertencias() {
   return (advertencias == null)? new ArrayList() : advertencias;
}

public void setProccessWarnings(boolean advertenciasProcesadas) {
   if (advertenciasProcesadas){
      advertencias = new ArrayList();
   }else{
      advertencias.clear();
      advertencias = null;
   }
   this.advertenciasProcesadas = advertenciasProcesadas;
}

public boolean isAdvertenciasProcesadas() {
   return this.advertenciasProcesadas;
}

/** Asegúrese de leer las advertencias antes de llamar a release() */
public void rellenarAdvertencias(SQLWarning SQLwarn){
   while (SQLwarn != null){
      ContenedorDatos cd = new ContenedorDatos(3);
      cd.setNombreCampoYValor(1, "message", SQLwarn.getMessage());
      cd.setNombreCampoYValor(2, "SQLState", SQLwarn.getSQLState());
      cd.setNombreCampoYValor(3, "vendorErrorCode", new Integer(SQLwarn.getErrorCode()));
      advertencias.add(cd);
      SQLwarn = SQLwarn.getNextWarning();
   }
}

//=========================================================================

public boolean isAlmacenStatements() {
   return almacenStatements;
}

public void borrarAlmacenStatement(){
   if (sentenciasAlmacenamiento != null){
      try{
         Iterator iter = sentenciasAlmacenamiento.iterator();
         while (iter.hasNext()){
            Object obj = iter.next();
            if (obj != null){
               ((Statement) obj).close();
            }
         }
         sentenciasAlmacenamiento.clear();
         contadorSentenciasAlmacenadas.clear();
         sentenciasId.clear();
         if (!almacenStatements){
```

```
            contadorSentenciasAlmacenadas = null;
            sentenciasAlmacenamiento = null;
            sentenciasId = null;
          }
        }catch (SQLException sqle){
          sqle.printStackTrace();
        }
      }
    }

    public void setAlmacenStatements(boolean almacenStatements){
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::==========
setAlmacenStatements() ============= almacenStatements: " + almacenStatements + " \n");
      if (almacenStatements){
        if (sentenciasAlmacenamiento == null){
          sentenciasAlmacenamiento = new ArrayList(maxLongitudAlmacenamientoSentencias);
          sentenciasId = new ArrayList(maxLongitudAlmacenamientoSentencias);
          contadorSentenciasAlmacenadas = new ArrayList(maxLongitudAlmacenamientoSentencias);
        }
        setCerrarStatementTrasEjecucion(false);
      }else{
        borrarAlmacenStatement();
      }
      this.almacenStatements = almacenStatements;
    }

    public int getMaxLongitudAlmacenamientoSentencias() {
      return this.maxLongitudAlmacenamientoSentencias;
    }

    public void setMaxLongitudAlmacenamientoSentencias(int maxLongitudAlmacenamientoSentencias) {
      this.maxLongitudAlmacenamientoSentencias = maxLongitudAlmacenamientoSentencias;
    }

    protected synchronized int addAlmacenStatement(String statementId, PreparedStatement statementParaAlmacenar) throws SQLException{
      if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::==========
addAlmacenStatement() ============= statementId: " + statementId + "::statementParaAlmacenar: " + statementParaAlmacenar + "\n");
      if (sentenciasId.isEmpty()){
        sentenciasId.add(statementId);
        sentenciasAlmacenamiento.add(statement);
        return 0;
      }
      int idx = sentenciasId.indexOf(statementId);
      if (idx < 0){
        if (sentenciasId.size() + 1 == maxLongitudAlmacenamientoSentencias){
          // Buscar y eliminar la list a menudo requerida para las sentencias (Statement)
          int iterCont = 0;
          int idxListaSentenciasUsadasFrecuentemente = 0;
          int listaSentenciasUsadasFrecuentemente = ((Integer) contadorSentenciasAlmacenadas.get(0)).intValue();
          Iterator iter = contadorSentenciasAlmacenadas.iterator();
          while (iter.hasNext()){
            int valorActual = ((Integer) iter.next()).intValue();
            if (valorActual < listaSentenciasUsadasFrecuentemente){
              listaSentenciasUsadasFrecuentemente = valorActual;
              idxListaSentenciasUsadasFrecuentemente = iterCont;
            }
            iterCont++;
          }
          sentenciasId.remove(idxListaSentenciasUsadasFrecuentemente);
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::addAlmacenStatement()::cerrando PreparedStatement: " + ((Statement)
sentenciasAlmacenamiento.get(idxListaSentenciasUsadasFrecuentemente)) + " \n");
          ((Statement) sentenciasAlmacenamiento.get(idxListaSentenciasUsadasFrecuentemente)).close();
          sentenciasAlmacenamiento.remove(idxListaSentenciasUsadasFrecuentemente);
          contadorSentenciasAlmacenadas.remove(idxListaSentenciasUsadasFrecuentemente);
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::==========
addCachedStatement()::remove ============= idxListaSentenciasUsadasFrecuentemente: " + idxListaSentenciasUsadasFrecuentemente + "\n");
        }
        sentenciasId.add(statementId);
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::==========
addAlmacenStatement()::statementParaAlmacenar: " + statementParaAlmacenar + " " + statementParaAlmacenar.getClass() + "\n");
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::==========
addAlmacenStatement()::statementParaAlmacenar instanceof PreparedStatement: " + (statementParaAlmacenar instanceof PreparedStatement)
+ "\n");
        if (statementParaAlmacenar instanceof PreparedStatement){
          sentenciasAlmacenamiento.add((PreparedStatement) statementParaAlmacenar);
        }else{
          sentenciasAlmacenamiento.add(statementParaAlmacenar);
        }
```

```
            idx = sentenciasId.size() - 1;
        } else{
            // la sentencia ya está en la cache, no hacer nada
        }
        return idx;
    }

    protected synchronized int addAlmacenStatement(String statementId, Statement statementParaAlmacenar) throws SQLException{
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
addAlmacenStatement() ============= statementId: " + statementId + "::statementParaAlmacenar: " + statementParaAlmacenar + "\n");
        if (sentenciasId.isEmpty()){
            sentenciasId.add(statementId);
            sentenciasAlmacenamiento.add(statement);
            return 0;
        }
        int idx = sentenciasId.indexOf(statementId);
        if (idx < 0){
            if (sentenciasId.size() + 1 == maxLongitudAlmacenamientoSentencias){
                // Buscar y eliminar la list a menudo requerida para las sentencias (Statement)
                int iterCont = 0;
                int idxListaSentenciasUsadasFrecuentemente = 0;
                int listaSentenciasUsadasFrecuentemente = ((Integer) contadorSentenciasAlmacenadas.get(0)).intValue();
                Iterator iter = contadorSentenciasAlmacenadas.iterator();
                while (iter.hasNext()){
                    int valorActual = ((Integer) iter.next()).intValue();
                    if (valorActual < listaSentenciasUsadasFrecuentemente){
                        listaSentenciasUsadasFrecuentemente = valorActual;
                        idxListaSentenciasUsadasFrecuentemente = iterCont;
                    }
                    iterCont++;
                }
                sentenciasId.remove(idxListaSentenciasUsadasFrecuentemente);
                if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::addAlmacenStatement()::cerrando Statement: " + ((Statement) sentenciasAlmacenamiento.get(idxListaSentenciasUsadasFrecuentemente)) + "
\n");
                ((Statement) sentenciasAlmacenamiento.get(idxListaSentenciasUsadasFrecuentemente)).close();
                sentenciasAlmacenamiento.remove(idxListaSentenciasUsadasFrecuentemente);
                contadorSentenciasAlmacenadas.remove(idxListaSentenciasUsadasFrecuentemente);
                if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
addAlmacenStatement()::remove ============= idxListaSentenciasUsadasFrecuentemente: " + idxListaSentenciasUsadasFrecuentemente + "\n");
            }
            sentenciasId.add(statementId);
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
addAlmacenStatement()::statementParaAlmacenar: " + statementParaAlmacenar + " " + statementParaAlmacenar.getClass() + "\n");
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::================
addAlmacenStatement()::statementParaAlmacenar instanceof PreparedStatement: " + (statementParaAlmacenar instanceof PreparedStatement)
+ "\n");
            if (statementParaAlmacenar instanceof PreparedStatement){
                sentenciasAlmacenamiento.add((PreparedStatement) statementParaAlmacenar);
            }else{
                sentenciasAlmacenamiento.add(statementParaAlmacenar);
            }
            idx = sentenciasId.size() - 1;
        } else{
            // la sentencia ya está en la cache, no hacer nada
        }
        return idx;
    }

    public String getCachedRowSetClassName() {
        return cachedRowSetClassName;
    }

    public void setCachedRowSetClassName(String cachedRowSetClassName) {
        this.cachedRowSetClassName = cachedRowSetClassName;
    }

    public void setCerrarResultSet(boolean cerrarResultSet) {
        this.cerrarResultSet = cerrarResultSet;
    }

    public boolean isCerrarResultSet(){
        return this.cerrarResultSet;
    }

    public String getNombreCursor() {
        return nombreCursor;
    }

    public void setNombreCursor(String nombreCursor) {
        this.nombreCursor = nombreCursor;
```

```
    }

    public char getProccessTransactionOnCloseConnection() {
        return proccessTransactionOnCloseConnection;
    }

    public void setProccessTransactionOnCloseConnection(char proccessTransactionOnCloseConnection) {
        this.proccessTransactionOnCloseConnection = proccessTransactionOnCloseConnection;
    }

    public Object getEJBContext() {
        return EJBContext;
    }

    public void setEJBContext(Object EJBContext) {
        this.EJBContext = EJBContext;
    }
}
```

Clase: FactoriaObjetoPersistente.java

```
package org.fpuoc;

import java.util.*;
import java.sql.*;
import java.lang.reflect.*;
import javax.sql.DataSource;

public abstract class FactoriaObjetoPersistente implements java.io.Serializable, Cloneable {

    public FactoriaObjetoPersistente() {
        if (Boolean.getBoolean("debug-fpuoc") || Boolean.getBoolean("debug-" + getClass().getName())){
            debug = true;
        }
    }

    public FactoriaObjetoPersistente(boolean synchronizedAccess){
        this.setSincronizacionAccesos(synchronizedAccess);
    }

    public FactoriaObjetoPersistente(Class requestor, boolean synchronizedAccess){
        this.requestor = requestor;
        this.setSincronizacionAccesos(synchronizedAccess);
    }

    public FactoriaObjetoPersistente(Class requestor) {
        this.requestor = requestor;
    }
    /*================================= inicio ======================*/
    protected boolean debug = false;

    protected Metadatos metadatos = null;
    /* Mapeo de las columnas de la Tabla con los campos de los objetos */
    protected Map mapping = null;
    /* Objeto para hacer persistente */
    protected Class requestor = null;
    /* Identificador de objeto */
    protected int objetoId = 0;
    /* Colección de objetos (data) */
    protected List listaObjetos = new ArrayList();
    /* Colección de objetos para eliminar en la BBDD */
    protected List objetosEliminados = new ArrayList();
    /* Se usa para enviar los objetos no modificados a otra capa */
    protected boolean descartarObjetosNoModificados;
    /* Asigna 'sequence' o deshabilita update/insert/delete */
    protected char[] datosModificacionSequence;
    public static final char DATOS_MODIFICACION_SEQUENCE_DELETE = 'D';
    public static final char DATOS_MODIFICACION_SEQUENCE_INSERT = 'I';
    public static final char DATOS_MODIFICACION_SEQUENCE_UPDATE = 'U';
    /* Deshabilita modificaciones en la BBDD */
    protected boolean soloLectura = false;

    /* Optimistic lock settings ***************************************/
    /* Genera procedimientos con WHERE usando solo la clave */
```

```
public static final char KEY_COLUMNS = 'K';
/* Genera procedimientos con WHERE con la clave y la columnas modificables */
public static final char KEY_AND_MODIFIED_COLUMNS = 'M';
/* Genera procedimientos con WHERE con al clave y todas columnas actualizables - este es
   el chequeo de concurrencia más severo */
public static final char KEY_AND_UPDATEBLE_COLUMNS = 'U';
/* Current Optimistic lock setting.
   Por defecto es KEY_COLUMNS (Solo Primary Key) */
protected char optimisticLock = KEY_COLUMNS;

/*
   Lanza la excepción si PreparedStatement.executeUpdate() no devuelve 1
   para cada ObjetoPersistente.
   Esto significa que es muy probable que la fila en la tabla de la BBDD
   fue modificada o eliminada por otro usuario / proceso.
 */
protected boolean throwOptimisticLockDeleteException = true;
protected boolean throwOptimisticLockUpdateException = true;
protected boolean throwFailedInsertException = true;

/* FIN de Optimistic lock settings *********************************/
/* Apuntador a catalogo en la BBDD */
protected String catalogo = null;
/* Si anularNombreCatalogo == true y catalogo en this.metadatos es igual a null entonces se usa catalogo como variable */
protected boolean anularNombreCatalogo = false;
/* Apuntador a esquema en la BBDD */
protected String esquema = null;
/* Apuntador a tabla en la BBDD */
protected String tabla = null;
/* Este objeto genera sentencias SQL para modificación de datos */
protected GrabarDatos grabarDatos = null;
/* Esta es una utilidad JDBC */
protected DBServicios dbServicios = null;
/* se usa como argumento en method.invoke(obj, EMPTY_READ_ARG)*/
protected Object[] EMPTY_READ_ARG = new Object[0];
/* usar Collections sincronizadas */
protected boolean sincronizacionAccesos = false;
/* Si lazyLoding == true entonces solo asignar valores originales vs. original y current */
protected boolean lazyLoading = false;
/* Registros por pagina */
protected int longitudPagina = 0;
/* No ejecutar INSERTS/UPDATES/DELETES - sólo generar el SQL y los parámetros */
protected boolean generarSoloSQL = false;
/* Controlar si cargar() añade objetos a listaObjetos */
protected boolean adjuntar = false;
/* Algunas BBDD fallan (como hsqdb 1.7.1) cuando la sentencia UPDATE se usa
 con TABLE_NAME.COLUMN_NAME colocada en la clausula SET. INSERT tambien falla */
protected boolean noNombreColumnaCompleto = false;
/* Lanza la excepción o ignora si ObjetoPersistente tien menos campos para mapear */
protected boolean throwMissingFieldException = false;
/* Se usa para determinar si create/deleted/updated ObjetoPersistente pertenece a este objeto factory */
protected long domainFactoryId = System.currentTimeMillis();
/* si no es null solo maneja excepciones */
protected ManejadorExcepciones manejadorExcepciones;
/* Si es 'false' suprime las excepciones cuando se llama a grabaDatos con listaObjetos vacía */
protected boolean exceptionListaObjetosVacia = false;
/* Indica que es un conveniente member/method */
protected boolean valido = true;
/* Crea y rellena Metadatos automaticamente */
protected boolean crearMetadatosDefault = true;
/* Lee un objeto desde ResultSet como un tipo de datos específico */
protected Transformador transformador = null;
/*================================= members end==========================*/

public int cargar(ResultSet[] rs) throws java.sql.SQLException{
   boolean origAppend = this.isAdjuntar();
   this.setAdjuntar(true);
   int contFilas = 0;
   if (debug) System.out.println("fpuoc @@@@@@@@@@@@@@ rs.length: " + rs.length);
   if (debug) System.out.println("fpuoc @@@@@@@@@@@@@@ rs: " + Arrays.asList(rs));

   for (int i = 0;i < rs.length;i++){
      if (debug) System.out.println("fpuoc @@@@@@@@@@@@@@ resultSet[" + i + "]: " + rs[i]);
      contFilas = contFilas + cargar(rs[i]);
      if (debug) System.out.println("fpuoc @@@@@@@@@@@@@@ rowCount_" + i + ": " + contFilas);
   }
   this.setAdjuntar(origAppend);
   return contFilas;
}

public int cargar(ResultSet rs) throws java.sql.SQLException{
```

```
        long ts1 = -1;
        long ts2 = -1;
        String nombreColum = null;
        String nombreCampo = null;
        ResultSetMetaData rsmd = rs.getMetaData();
        int rsContColum = rsmd.getColumnCount();
        List rsColums = new ArrayList(rsContColum);
        for (int i = 1;i <= rsContColum;i++){
           rsColums.add(rsmd.getColumnName(i));
        }
        int contColum = 0;
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::load()::requestor class::" +
requestor.getName());
        if (!isAdjuntar()){
           this.listaObjetos.clear();
           if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::load()::objetoId 1: " +
objetoId);
           this.objetoId = 0;
           if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() +"::load()::objetoId 2: " + objetoId);
        }
        try{
           if (debug) ts1 = System.currentTimeMillis();
           if (this.metadatos == null){
              rellenarMetadatos();
              if (debug) ts2 = System.currentTimeMillis();
              if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::rellenarMetadatos: " +
(ts2 - ts1) + " ms");
           }
           contColum = this.metadatos.getContadorColumna();
           if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::load()::columnCount: " +
contColum);
           if (transformador == null){
              transformador = new TransformadorDefecto(rs);
           }else{
              transformador.setResultSet(rs);
           }
           while (rs.next()){
              ContenedorDatos valoresOriginales = null;
              if (!isSoloLectura()){
                 valoresOriginales = new ContenedorDatos(contColum);
              }
              Object op = requestor.newInstance();
              if (!isLazyLoading()){
                 ((ObjetoPersistente) op).setCargando(true);
              }
              for (int idxColum = 1; idxColum <= contColum; idxColum++){
                 nombreCampo = this.metadatos.getNombreCampo(idxColum);
                 if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::load()::fieldName: " +
nombreCampo);
                 if (nombreCampo != null){  // ResultSet may have more columns then this Object fields, so skip unneeded column
                    nombreColum = this.metadatos.getNombreColumna(idxColum);
                    if (!rsColums.contains(nombreColum)){
                       // ResultSet has less columns then ObjetoPersistente fields
                       this.metadatos.setGrabable(idxColum, false);
                       continue;
                    }
                    String fieldClassName = this.metadatos.getCampoNombreClass(idxColum);
                    Object[] convertedValue = transformador.transformar(fieldClassName, nombreColum);
                    if (!isSoloLectura()){
                       valoresOriginales.setNombreCampoYValor(idxColum, nombreCampo, convertedValue[0]);
                    }
                    if (!isLazyLoading() || isSoloLectura()){
                       this.metadatos.getGrabarMethod(idxColum).invoke(op, convertedValue);
                    }
                 }
                 if (debug) System.out.print("\n");
              }
              if (!isSoloLectura()){
                 ((ObjetoPersistente) op).setValoresOriginales(valoresOriginales);
              }
              if (!isLazyLoading()){
                 ((ObjetoPersistente) op).setCargando(false);
                 ((ObjetoPersistente) op).setCargado(true);
              }
              if (isSincronizacionAccesos()){
                 ((ObjetoPersistente) op).setObjetoId(this.getSyncNextObjetoId());
              }else{
                 ((ObjetoPersistente) op).setObjetoId(this.getNextObjetoId());
              }
```

```
            ((ObjetoPersistente) op).setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
            ((ObjetoPersistente) op).setPersistenteFactoryId(domainFactoryId);
            this.listaObjetos.add(op);
          }
        }catch (InstantiationException ie){
          ie.printStackTrace();
          throw new java.sql.SQLException("Cannot Instantiate: " + requestor);
        }catch (IllegalAccessException iae){
          iae.printStackTrace();
          throw new java.sql.SQLException(iae.getMessage());
        }catch (InvocationTargetException ite){
          ite.printStackTrace();
          throw new java.sql.SQLException(ite.getTargetException().getMessage());
        }catch (Exception e){
          e.printStackTrace();
          throw new java.sql.SQLException(e.getMessage());
        }
        ts2 = System.currentTimeMillis();
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::load() total: " + (ts2 - ts1 + "
ms"));
        return this.listaObjetos.size();
    }

    /** Leer la propiedad de mapping.
     * @return Value of property mapping.
     *
     */
    public Map getMapping() {
        return mapping;
    }

    /** Asignar nueva propiedad a mapping.
     * @param mapping New obj of property mapping.
     *
     */
    public void setMapping(Map mapping) {
        this.mapping = mapping;
    }

    public void setMapping(String mapping) {
        StringTokenizer tokenizer = new StringTokenizer(mapping, ",", false);
        if (this.mapping == null){
          this.mapping = new Properties();
        }else{
          this.mapping.clear();
        }
        while (tokenizer.hasMoreTokens()){
          String token = tokenizer.nextToken();
          int idx = token.indexOf("=");
          if (idx == -1){
            throw new IllegalArgumentException("FactoriaObjetoPersistente::setMapping(String mapping): " + mapping + "(Formato Arg.:
<COLUMN_NAME>=<fieldName>)");
          }
          this.mapping.put(token.substring(0, idx).trim(), token.substring(idx + 1).trim());
        }
    }

    /** Leer la propiedad de requestor.
     * @return Value of property requestor.
     *
     */
    public Class getRequestor() {
        return requestor;
    }

    /** Asignar nueva propiedad a requestor.
     * @param requestor New obj of property requestor.
     *
     */
    public void setRequestor(Class requestor) {
        this.requestor = requestor;
    }

    public List getPrimaryKey(String catalogo, String esquema, String tabla) throws SQLException{
        Object pk = PKCache.getInstance().getPrimaryKey(catalogo, esquema, tabla);
        if (pk == null){
          pk = PKCache.getInstance().getPrimaryKey(this.dbServicios.getConnection(), catalogo, esquema, tabla);
        }
        return (List) pk;
    }
```

```
public void setPrimaryKey(String catalogo, String esquema, String tabla, List pk) {
   PKCache.getInstance().setPrimaryKey(catalogo, esquema, tabla, pk);
}

public int setObjeto(ObjetoPersistente op){
   int idx = -1;
   int id = 0;
   if (op.getPersistenteFactoryId() != domainFactoryId){
      op.setPersistenteFactoryId(domainFactoryId);
      if (isSincronizacionAccesos()){
         id = this.getSyncNextObjetoId();
         op.setObjetoId(id);
      }else{
         id = this.getNextObjetoId();
         op.setObjetoId(id);
      }
      this.listaObjetos.add(op);
      idx = this.listaObjetos.size() + 1;
   }else{
      id = op.getObjetoId();
      if (id == 0){  // new object
         if (isSincronizacionAccesos()){
            id = this.getSyncNextObjetoId();
            op.setObjetoId(id);
         }else{
            id = this.getNextObjetoId();
            op.setObjetoId(id);
         }
         this.listaObjetos.add(op);
         idx = this.listaObjetos.size() + 1;
      }else{
         idx = findIdxUsandoObjetoId(id);
         if (idx > 0){
            this.listaObjetos.set(idx - 1, op);
         }else{
            this.listaObjetos.add(op);
         }
      }
   }
   return idx;
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... */
public void setObjeto(int idx, ObjetoPersistente op){
   int id = 0;
   if (op.getPersistenteFactoryId() != domainFactoryId){
      op.setPersistenteFactoryId(domainFactoryId);
      if (isSincronizacionAccesos()){
         id = this.getSyncNextObjetoId();
         op.setObjetoId(id);
      }else{
         id = this.getNextObjetoId();
         op.setObjetoId(id);
      }
   }else{
      if (op.getObjetoId() == 0){
         if (isSincronizacionAccesos()){
            id = this.getSyncNextObjetoId();
            op.setObjetoId(id);
         }else{
            id = this.getNextObjetoId();
            op.setObjetoId(id);
         }
      }
   }
   this.listaObjetos.set(idx - 1, op);
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... */
public void addObjeto(int idx, ObjetoPersistente op){
   int id = 0;
   if (op.getPersistenteFactoryId() != domainFactoryId){
      op.setPersistenteFactoryId(domainFactoryId);
      if (isSincronizacionAccesos()){
         id = this.getSyncNextObjetoId();
         op.setObjetoId(id);
      }else{
         id = this.getNextObjetoId();
         op.setObjetoId(id);
```

```
        }
    }else{
        if (op.getObjetoId() == 0){
            if (isSincronizacionAccesos()){
                id = this.getSyncNextObjetoId();
                op.setObjetoId(id);
            }else{
                id = this.getNextObjetoId();
                op.setObjetoId(id);
            }
        }
    }
    this.listaObjetos.add(idx - 1, op);
}

public int addObjeto(ObjetoPersistente op){
    int id = 0;
    if (op.getPersistenteFactoryId() != domainFactoryId){
        op.setPersistenteFactoryId(domainFactoryId);
        if (isSincronizacionAccesos()){
            id = this.getSyncNextObjetoId();
            op.setObjetoId(id);
        }else{
            id = this.getNextObjetoId();
            op.setObjetoId(id);
        }
    }else{
        if (op.getObjetoId() == 0){
            if (isSincronizacionAccesos()){
                id = this.getSyncNextObjetoId();
                op.setObjetoId(id);
            }else{
                id = this.getNextObjetoId();
                op.setObjetoId(id);
            }
        }
    }
    this.listaObjetos.add(op);
    return this.listaObjetos.size() - 1;
}

public ObjetoPersistente findObjetoUsandoObjetoId(int objetoId){
    int idx = findIdxUsandoObjetoId(objetoId);
    return (ObjetoPersistente) this.listaObjetos.get(idx - 1);
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... */
public int findIdxUsandoObjetoId(int objetoId){
    int id = 0;
    int idx = 0;
    Iterator it = this.listaObjetos.iterator();
    while (it.hasNext()){
        idx ++;
        ObjetoPersistente op = (ObjetoPersistente) it.next();
        id = op.getObjetoId();
        if (objetoId == id){
            break;
        }
    }
    return idx;
}

protected synchronized int getSyncNextObjetoId() {
    this.objetoId ++;
    return this.objetoId;
}

protected int getNextObjetoId() {
    this.objetoId ++;
    return this.objetoId;
}

/** Leer la propiedad de listaObjetos.
 * @return Value of property listaObjetos.
 *
 */
public List getListaObjetos() {
    return listaObjetos;
}

/** Asignar nueva propiedad a listaObjetos.
```

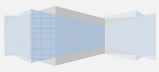
```
 * @param listaObjetos New obj of property listaObjetos.
 *
 */
public void setListaObjetos(List listaObjetos) {
   this.listaObjetos = listaObjetos;
}

/* No se elimina de la BBDD solo de listaObjetos */
public boolean descartar(ObjetoPersistente op) {
   int idx = findIdxUsandoObjetoId(op.getObjetoId());
   if (idx > 0){
      this.listaObjetos.remove(idx - 1);
      return true;
   }
   return false;
}

public boolean eliminar(ObjetoPersistente op){
   boolean completado = false;
   int removedIdx = -1;
   if (op.getPersistenteFactoryId() == domainFactoryId){
      int idx = findIdxUsandoObjetoId(op.getObjetoId());
      if (!antesEliminar(op)){
         return completado;
      }
      if (idx > 0){
         if (op.getEstadoPersistencia() == ObjetoPersistente.CREADO){
            this.listaObjetos.remove(idx - 1); // No se elimina de la BBDD solo de listaObjetos
            completado = false;
         }else{
            if (op.eliminar()){
               this.objetosEliminados.add(op);
               removedIdx = this.objetosEliminados.size() + 1;
               this.listaObjetos.remove(idx - 1);
               completado = true;
            }else{ // descartar?
               //this.listaObjetos.eliminar(idx - 1);
               completado = false;
            }
         }
      }else{
         if (isSincronizacionAccesos()){
            op.setObjetoId(this.getSyncNextObjetoId());
         }else{
            op.setObjetoId(this.getNextObjetoId());
         }
         if (op.getEstadoPersistencia() == ObjetoPersistente.CREADO){
            // No se elimina de la BBDD solo de listaObjetos
            completado = false;
         }else{
            if (op.eliminar()){
               this.objetosEliminados.add(op);
               completado = true;
            }
         }
      }
      if (!despuesEliminar(removedIdx)){
         return false;
      }
   }else{
      op.setPersistenteFactoryId(domainFactoryId);
      if (!antesEliminar(op)){
         return false;
      }
      if (isSincronizacionAccesos()){
         op.setObjetoId(this.getSyncNextObjetoId());
      }else{
         op.setObjetoId(this.getNextObjetoId());
      }

      if (op.getEstadoPersistencia() == ObjetoPersistente.CREADO){
         this.objetosEliminados.add(op); // No se elimina de la BBDD solo de listaObjetos
         completado = false;
      }else{
         if (op.eliminar()){
            this.objetosEliminados.add(op);
            completado = true;
         }
      }
```

```
      if (!despuesEliminar(removedIdx)){
        return false;
      }
    }
    return completado;
}

// es necesario contar con este paso
public boolean antesEliminar(ObjetoPersistente op) {
    return true;
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... es necesario contar con este paso */
public boolean despuesEliminar(int idx){
    return true;
}

public boolean almacenar(ObjetoPersistente op) {
    boolean completado = false;
    int idx = -1;
    if (op.getPersistenteFactoryId() == domainFactoryId){  // op podía ser de otro FactoriaObjetoPersistente o creado a mano o de otra manera
      idx = findIdxUsandoObjetoId(op.getObjetoId());
      if (!antesAlmacenar(op)){
        return false;
      }
      if (idx > 0){
        if (op.almacenar()){
          this.listaObjetos.set(idx - 1, op);
          completado = true;
        }
      }else{
        if (isSincronizacionAccesos()){
          op.setObjetoId(this.getSyncNextObjetoId());
        }else{
          op.setObjetoId(this.getNextObjetoId());
        }
        if (op.almacenar()){
          this.listaObjetos.add(op);
          idx = this.listaObjetos.size() - 1;
          completado = true;
        }else{
          completado = false;
        }
      }
      if (!despuesAlmacenar(idx)){
        return false;
      }
    }else{
      op.setPersistenteFactoryId(domainFactoryId);
      if (isSincronizacionAccesos()){
        op.setObjetoId(this.getSyncNextObjetoId());
      }else{
        op.setObjetoId(this.getNextObjetoId());
      }
      if (!antesAlmacenar(op)){
        return false;
      }
      if (op.almacenar()){
        this.listaObjetos.add(op);
        idx = this.listaObjetos.size() - 1;
        completado = true;
      }else{
        completado = false;
      }
      if (!despuesAlmacenar(idx)){
        return false;
      }
    }
    return completado;
}

// es necesario contar con este paso
public boolean antesAlmacenar(ObjetoPersistente domainObjectx){
    return true;
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... es necesario contar con este paso */
public boolean despuesAlmacenar(int idx){
    return true;
}
```

```
public boolean crear(ObjetoPersistente op) {
   if (op.getPersistenteFactoryId() != domainFactoryId){
      op.setPersistenteFactoryId(domainFactoryId);
   }
   int idx = -1;
   boolean completado = false;
   if (!antesCrear(op)){
      return false;
   }
   if (op.crear()){
      idx = this.setObjeto(op);
      completado = true;
   }
   if (!despuesCrear(idx)){
      return false;
   }
   return completado;
}

// es necesario contar con este paso
public boolean antesCrear(ObjetoPersistente domainObject){
   return true;
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... es necesario contar con este paso */
public boolean despuesCrear(int idx){
   return true;
}

/** Leer la propiedad de objetosEliminados.
 * @return Value of property objetosEliminados.
 *
 */
public List getObjetosEliminados() {
   return objetosEliminados;
}

/** Asignar nueva propiedad a objetosEliminados.
 * @param objetosEliminados New obj of property objetosEliminados.
 *
 */
public void setObjetosEliminados(List objetosEliminados) {
   this.objetosEliminados = objetosEliminados;
}

/** Indexed Leer la propiedad de datosModificacionSequence.
 * @param idx Index of the property.
 * @return Value of the property at <CODE>idx</CODE>.
 *
 */
public char getDatosModificacionSequence(int idx) {
   return this.datosModificacionSequence[idx];
}

/** Leer la propiedad de datosModificacionSequence.
 * @return Value of property datosModificacionSequence.
 *
 */
public char[] getDatosModificacionSequence() {
   return this.datosModificacionSequence;
}



/** Asignar nueva propiedad a datosModificacionSequence.
 * @param datosModificacionSequence New obj of property datosModificacionSequence.
 *
 */
public void setDatosModificacionSequence(char[] datosModificacionSequence) {
   this.datosModificacionSequence = datosModificacionSequence;
}

/** Leer la propiedad de descartarObjetosNoModificados.
 * @return Value of property descartarObjetosNoModificados.
 *
 */
public boolean isDescartarObjetosNoModificados() {
   return descartarObjetosNoModificados;
}
```

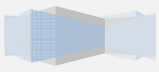
```
/** Asignar nueva propiedad a descartarObjetosNoModificados.
 * @param descartarObjetosNoModificados New obj of property descartarObjetosNoModificados.
 *
 */
public void setDescartarObjetosNoModificados(boolean descartarObjetosNoModificados) {
   ListIterator li = this.listaObjetos.listIterator();
   while (li.hasNext()){
      ObjetoPersistente op = (ObjetoPersistente) li.next();
      if (op.getEstadoPersistencia() == op.ORIGINAL){
         li.remove();
      }
   }
   this.descartarObjetosNoModificados = descartarObjetosNoModificados;
}

/** Leer la propiedad de optimisticLock.
 * @return Value of property optimisticLock.
 *
 */
public char getOptimisticLock() {
   return optimisticLock;
}

/** Asignar nueva propiedad a optimisticLock.
 * @param optimisticLock New obj of property optimisticLock.
 *
 */
public void setOptimisticLock(char optimisticLock) {
   this.optimisticLock = optimisticLock;
}

/** Leer la propiedad de soloLectura.
 * @return Value of property soloLectura.
 *
 */
public boolean isSoloLectura() {
   return soloLectura;
}

/** Asignar nueva propiedad a soloLectura.
 * @param soloLectura New obj of property soloLectura.
 * makes this non-updateable.
 */
public void setSoloLectura(boolean soloLectura) {
   this.soloLectura = soloLectura;
}

public void setOriginal() {
   if (isSoloLectura()){
      throw new IllegalStateException("No se puede usar setOriginal() para objetos readOnly");
   }
   listaObjetos.add(this.objetosEliminados);
   ListIterator li = listaObjetos.listIterator();
   while (li.hasNext()){
      ObjetoPersistente op = (ObjetoPersistente) li.next();
      if (op.getEstadoPersistencia() == ObjetoPersistente.ELIMINADO &&
            op.getValoresOriginales() != null &&
            op.getValoresOriginales().getContadorCampos() > 0){
         op.setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
      }else if (op.getEstadoPersistencia() == ObjetoPersistente.CREADO &&
            op.getValoresOriginales() == null ||
            op.getValoresOriginales().getContadorCampos() < 1){
         li.remove();
      }else if (op.getEstadoPersistencia() == ObjetoPersistente.ALMACENADO &&
            op.getValoresOriginales() != null &&
            op.getValoresOriginales().getContadorCampos() > 0){
         op.setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
      }else{
         throw new IllegalArgumentException("No valido estado de SQL");
      }
   }
   objetosEliminados.clear();
}

/** Leer la propiedad de esquema.
 * @return Value of property esquema.
 *
 */
public String getEsquema() {
   return esquema;
```

```
  }

  /** Asignar nueva propiedad a esquema.
   * @param esquema New obj of property esquema.
   *
   */
  public void setEsquema(String esquema) {
     this.esquema = esquema;
  }

  /** Leer la propiedad de tabla.
   * @return Value of property tabla.
   *
   */
  public String getTabla() {
     return tabla;
  }

  /** Asignar nueva propiedad a tabla.
   * @param tabla New obj of property tabla.
   *
   */
  public void setTabla(String tabla) {
     this.tabla = tabla;
  }

  public boolean grabarDatos(){
     if (!isValido()){
        return false;
     }
     if (!antesGrabarDatos()){
        return false;
     }
     if (this.grabarDatos == null){
        this.grabarDatos = new DBGrabarDatos();
     }
     this.grabarDatos.setExcepcionEnObjectListVacios(exceptionListaObjetosVacia);
     boolean completado = this.grabarDatos.grabaDatos(this);
     if (!despuesGrabarDatos()){
        return false;
     }
     return completado;
  }

  // es necesario contar con este paso
  public boolean antesGrabarDatos(){
     return true;
  }

  // es necesario contar con este paso
  public boolean despuesGrabarDatos(){
     return true;
  }

  /** Leer la propiedad de metadatos.
   * @return Value of property metadatos.
   *
   */

  public Metadatos getMetadatos() {
     if (crearMetadatosDefault && this.metadatos == null){
        rellenarMetadatos();
     }
     return this.metadatos;
  }

  public void rellenarMetadatos() {
     try{
        this.metadatos = new Metadatos();
        this.metadatos.setThrowExcepcionCampoAusente(throwMissingFieldException);
        this.metadatos.llenar(getMapping(), this.requestor);
        Collection tablas = (this.metadatos.getTablasMap()).values();
        if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::metadatos.getTables(): " +
this.metadatos.getTablasMap());
        if (!this.isSoloLectura()){ // no es necesario PK si es tipo ReadOnly
           /* Pide todos los nombre de tabla distintos por solicitud (ObjetoPersistente) */
           Iterator iter = tablas.iterator();
           while (iter.hasNext()){
              String[] tablaId = (String[]) iter.next();
```

```
            if (tablaId[0] == null && anularNombreCatalogo){
               tablaId[0] = this.catalogo;
            }
            List pk = getPrimaryKey(tablaId[0], tablaId[1], tablaId[2]);
            if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::pk: " + pk);
          }
        }
        if (tablas.size() == 1){
          //hay una sola tabla para este objeto, de otra manera se debe asignar catalogo y/o esquema y tabla para update/delete/insert cada tabla
          if (this.metadatos.getNombreCatalogo(1) == null && !anularNombreCatalogo){
            setCatalogo(this.catalogo);
          }
          setEsquema(this.metadatos.getNombreEsquema(1));
          setTabla(this.metadatos.getNombreTabla(1));
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::this.catalogo: " +
this.catalogo);
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::esquema: " +
getEsquema());
          if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::tablas: " + getTabla());
        }
      }catch (Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
      }
  }

  /** Asignar nueva propiedad a metadatos.
   * @param metadatos New obj of property metadatos.
   *
   */
  public void setMetadatos(Metadatos metadatos) {
    this.metadatos = metadatos;
  }

  /** Leer la propiedad de dbServicios.
   * @return Value of property dbServicios.
   *
   */
  public DBServicios getDBServicios() {
    if (this.dbServicios == null){
      this.dbServicios = new DBServicios();
    }
    return this.dbServicios;
  }

  /** Asignar nueva propiedad a dbServicios.
   * @param dbServicios New obj of property dbServicios.
   *
   */
  public void setDBServicios(DBServicios dbServicios) {
    this.dbServicios = dbServicios;
  }

  /** Leer la propiedad de throwOptimisticLockDeleteException.
   * @return Value of property throwOptimisticLockDeleteException.
   *
   */
  public boolean isThrowOptimisticLockDeleteException() {
    return throwOptimisticLockDeleteException;
  }

  /** Asignar nueva propiedad a throwOptimisticLockDeleteException.
   * @param throwOptimisticLockDeleteException New obj of property throwOptimisticLockDeleteException.
   *
   */
  public void setThrowOptimisticLockDeleteException(boolean throwOptimisticLockDeleteException) {
    this.throwOptimisticLockDeleteException = throwOptimisticLockDeleteException;
  }

  /** Leer la propiedad de throwOptimisticLockUpdateException.
   * @return Value of property throwOptimisticLockUpdateException.
   *
   */
  public boolean isThrowOptimisticLockUpdateException() {
    return throwOptimisticLockUpdateException;
  }

  /** Asignar nueva propiedad a throwOptimisticLockUpdateException.
   * @param throwOptimisticLockUpdateException New obj of property throwOptimisticLockUpdateException.
   *
   */
```

```
  public void setThrowOptimisticLockUpdateException(boolean throwOptimisticLockUpdateException) {
     this.throwOptimisticLockUpdateException = throwOptimisticLockUpdateException;
  }

  public void setThrowFailedInsertException(boolean throwFailedInsertException){
     this.throwFailedInsertException = throwFailedInsertException;
  }

  public boolean isThrowFailedInsertException(){
     return this.throwFailedInsertException;
  }

  /**
   * MUY IMPORTANTE! ************************************************************
   * Se debe llamar a este método tras una llamada COMMIT satisfactorio a fin de
   * conseguir sincronizar los valores originales y los actuales: sobreescribe los
   * valores actuales a los originales y cambia el estado de la variable
   * persistentState == ORIGINAL y descarta los objetos borrados de la BBDD
   */
  public void sincronizarEstadoPersistencia(){
     if (isSoloLectura()){
        throw new IllegalStateException("Cannot call sincronizarEstadoPersistencia() for readOnly object");
     }
     try{
        objetosEliminados.clear();
        ListIterator li = listaObjetos.listIterator();
        while (li.hasNext()){
           boolean copiarValores = true;
           boolean creado = false;
           ObjetoPersistente op = (ObjetoPersistente) li.next();
           if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this
+"::syncSqlStatus()::ObjetoPersistente: " + op + " \n");
           if (op.getEstadoPersistencia() == ObjetoPersistente.ELIMINADO &&
                 op.getValoresOriginales() != null &&
                 op.getValoresOriginales().getContadorCampos() > 0){
              li.remove();
              copiarValores = false;
           }else if (op.getEstadoPersistencia() == ObjetoPersistente.CREADO &&
                 (op.getValoresOriginales() == null ||
                 op.getValoresOriginales().getContadorCampos() < 1)){
              op.setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
              creado = true;
           }else if (op.getEstadoPersistencia() == ObjetoPersistente.ALMACENADO &&
                 op.getValoresOriginales() != null &&
                 op.getValoresOriginales().getContadorCampos() > 0){
              op.setEstadoPersistencia(ObjetoPersistente.ORIGINAL);
           }else if (op.getEstadoPersistencia() == ObjetoPersistente.ORIGINAL){
              // no hacer nada
              continue;
           }else{
              throw new IllegalArgumentException("Invalid PersistentState");
           }
           if (copiarValores){ // Sincroniza values en el Objeto y sus valoresOriginales
              ContenedorDatos valoresOriginales = null;
              if (creado){
                 valoresOriginales = new ContenedorDatos(mapping.size());
              }else{
                 valoresOriginales = op.getValoresOriginales();
              }
              Iterator iter = mapping.values().iterator();
              int idxCampo = 0;
              if (debug) System.out.println("fpuoc ============ " + new java.util.Date() + " " + this.getClass() + "::" + this +"::syncSqlStatus()::creado:
" + creado + " \n");
              while(iter.hasNext()){
                 idxCampo ++;
                 String nombreCampo  = (String) iter.next();
                 if (creado){
                    Method leeMethod = this.metadatos.getLeerMethod(this.metadatos.getIndiceColumnaPorNombreCampo(nombreCampo));
                    Object obj = leeMethod.invoke(op, EMPTY_READ_ARG);
                    valoresOriginales.setNombreCampoYValor(idxCampo, nombreCampo, obj);
                 }else{
                    if (valoresOriginales.findIndiceCampo(nombreCampo) != -1){
                       Method leeMethod = this.metadatos.getLeerMethod(this.metadatos.getIndiceColumnaPorNombreCampo(nombreCampo));
                       Object obj = leeMethod.invoke(op, EMPTY_READ_ARG);
                       op.setValorOriginal(nombreCampo, obj);
                    }
                 }
              }
              if (creado){
```

```
                op.setValoresOriginales(valoresOriginales);
                creado = false;
              }
              op.setModificado(false);
            }
          }
          if (this.grabarDatos != null){
            this.grabarDatos.reset();
          }
          valido = true;
        }catch (Throwable t){
          if (t instanceof InvocationTargetException){
            throw new RuntimeException(((InvocationTargetException) t).getTargetException());
          }else{
            throw new RuntimeException(t);
          }
        }
      }

      /** Leer la propiedad de removedCount.
       * @return Value of property removedCount.
       *
       */
      public int getContadorBajas() {
        if (grabarDatos == null){
          throw new NullPointerException("DataWriter no inicializado");
        }
        return grabarDatos.getContadorBajas();
      }

      /** Leer la propiedad de createdCount.
       * @return Value of property createdCount.
       *
       */
      public int getContadorAltas() {
        if (grabarDatos == null){
          throw new NullPointerException("DataWriter no inicializado");
        }
        return grabarDatos.getContadorAltas();
      }

      /** Leer la propiedad de modifiedCount.
       * @return Value of property modifiedCount.
       *
       */
      public int getContadorModificaciones() {
        if (grabarDatos == null){
          throw new NullPointerException("DataWriter no inicializado");
        }
        return grabarDatos.getContadorModificaciones();
      }

      /** Leer la propiedad de grabarDatos.
       * @return Value of property grabarDatos.
       *
       */
      public GrabarDatos getGrabarDatos() {
        if (grabarDatos == null){
          grabarDatos = new DBGrabarDatos();
        }
        return grabarDatos;
      }

      /** Asignar nueva propiedad a grabarDatos.
       * @param grabarDatos New obj of property grabarDatos.
       *
       */
      public void setGrabarDatos(GrabarDatos grabarDatos) {
        this.grabarDatos = grabarDatos;
      }

      /** Leer la propiedad de sincronizacionAccesos.
       * @return Value of property sincronizacionAccesos.
       *
       */
      public boolean isSincronizacionAccesos() {
        return sincronizacionAccesos;
      }

      /** Asignar nueva propiedad a sincronizacionAccesos.
       * @param sincronizacionAccesos New obj of property sincronizacionAccesos.
```

```
 *
 */
public void setSincronizacionAccesos(boolean sincronizacionAccesos) {
   if (sincronizacionAccesos){
      this.objetosEliminados = Collections.synchronizedList(objetosEliminados);
      this.listaObjetos = Collections.synchronizedList(listaObjetos);
      this.sincronizacionAccesos = sincronizacionAccesos;
   }
}

/** Leer la propiedad de lazyLoading.
 * @return Value of property lazyLoading.
 *
 */
public boolean isLazyLoading() {
   return lazyLoading;
}

/** Asignar nueva propiedad a lazyLoading.
 * @param lazyLoading New obj of property lazyLoading.
 *
 */
public void setLazyLoading(boolean lazyLoading) {
   this.lazyLoading = lazyLoading;
}

/** Leer la propiedad de longitudPagina.
 * @return Value of property longitudPagina.
 *
 */
public int getLongitudPagina() {
   return longitudPagina;
}

/** Asignar nueva propiedad a longitudPagina.
 * @param longitudPagina New obj of property longitudPagina.
 *
 */
public void setLongitudPagina(int longitudPagina) {
   if (longitudPagina < 0){
      throw new IllegalArgumentException("Longitud pagina no valida: " + longitudPagina);
   }
   this.longitudPagina = longitudPagina;
}

/** No puede ser cero el valor del indice idx: idx del primer objeto es 1, del segundo es 2, etc... */
public MarcadorPagina getPagina(int numeroPagina) {
   MarcadorPagina mp = new MarcadorPagina();
   mp.setObjetosTotales(this.listaObjetos.size());
   if (getLongitudPagina() == 0){
      mp.setNumeroPagina(1);
      mp.setTotalPaginas(1);
      mp.setPagina(this.listaObjetos);
   }else{
      List pagina = new ArrayList();
      if (this.listaObjetos.isEmpty()){
         mp.setNumeroPagina(0);
         mp.setTotalPaginas(0);
         mp.setPagina(pagina);
      }else{
         double d1 = new Integer(this.listaObjetos.size()).doubleValue();
         double d2 = new Integer(this.longitudPagina).doubleValue();
         double d3 = Math.ceil(d1 / d2);
         int totalPaginas = (new Double(d3)).intValue();
         mp.setTotalPaginas(totalPaginas);
         if (numeroPagina > totalPaginas){
            numeroPagina = totalPaginas;
         }else if (numeroPagina < 1){
            numeroPagina = 1;
         }else{
            //
         }
         mp.setNumeroPagina(numeroPagina);
         ListIterator li = this.listaObjetos.listIterator((numeroPagina - 1) * getLongitudPagina());
         int cont = 0;
         while (li.hasNext() && cont < getLongitudPagina()){
            Object obj = li.next();
            pagina.add(obj);
            cont ++;
```

```
            }
            mp.setPagina(pagina);
        }
    }
    return mp;
}

/** Leer la propiedad de generarSoloSQL.
 * @return Value of property generarSoloSQL.
 *
 */
public boolean isGenerarSoloSQL() {
    return generarSoloSQL;
}

/** Asignar nueva propiedad a generarSoloSQL.
 * @param generarSoloSQL New obj of property generarSoloSQL.
 *
 */
public void setGenerarSoloSQL(boolean generarSoloSQL) {
    this.generarSoloSQL = generarSoloSQL;
}

public List getSQLGeneradas() {
    if (grabarDatos == null){
        throw new NullPointerException("DataWriter no inicializado");
    }
    return this.grabarDatos.getSQLGeneradas();
}

public String getSQLcomoXMLGeneradas() {
    if (grabarDatos == null){
        throw new NullPointerException("DataWriter no inicializado");
    }
    return this.grabarDatos.getGenerarSentenciasSQLcomoXML();
}

/** Leer la propiedad de adjuntar.
 * @return Value of property adjuntar.
 *
 */
public boolean isAdjuntar() {
    return adjuntar;
}

/** Asignar nueva propiedad a adjuntar.
 * @param adjuntar New obj of property adjuntar.
 *
 */
public void setAdjuntar(boolean adjuntar) {
    this.adjuntar = adjuntar;
}

public Throwable getErroresDetectados() {
    if (this.grabarDatos == null){
        return null;
    }
    return this.grabarDatos.getErroresDetectados();
}

/** Leer la propiedad de catalogo.
 * @return Value of property catalogo.
 *
 */
public String getCatalogo() {
    return catalogo;
}

/** Asignar nueva propiedad a catalogo.
 * @param catalogo New obj of property catalogo.
 *
 */
public void setCatalogo(String catalogo) {
    this.catalogo = catalogo;
}

/** Leer la propiedad de noNombreColumnaCompleto.
 * @return Value of property noNombreColumnaCompleto.
 *
 */
public boolean isNoNombreColumnaCompleto() {
```

```
    return noNombreColumnaCompleto;
}

/** Asignar nueva propiedad a noNombreColumnaCompleto.
 * @param noNombreColumnaCompleto New obj of property noNombreColumnaCompleto.
 *
 */
public void setNoNombreColumnaCompleto(boolean noNombreColumnaCompleto) {
    this.noNombreColumnaCompleto = noNombreColumnaCompleto;
}

/**
 * Leer la propiedad de debug.
 * @return Value of property debug.
 */
public boolean isDebug() {
    return debug;
}

public void limpiarDatos(){
    if (this.listaObjetos != null) this.listaObjetos.clear();
    if (this.objetosEliminados != null) this.objetosEliminados.clear();
    valido = true;
}

/**
 * Asignar nueva propiedad a debug.
 * @param debug New obj of property debug.
 */
public void setDebug(boolean debug) {
    this.debug = debug;
}

public boolean isAnularNombreCatalogo() {
    return anularNombreCatalogo;
}

public void setAnularNombreCatalogo(boolean anularNombreCatalogo) {
    this.anularNombreCatalogo = anularNombreCatalogo;
}

public void setManejadorExcepciones(ManejadorExcepciones manejadorExcepciones){
    this.manejadorExcepciones = manejadorExcepciones;
    manejadorExcepciones.setFactoriaObjetoPersistente(this);
}

public ManejadorExcepciones getManejadorExcepciones(){
    return this.manejadorExcepciones;
}

// es necesario contar con este paso
public boolean isValido() {
    return valido;
}

// crear ContenedorDatos vacío desde Metadatos
public ContenedorDatos crearValoresOriginales() throws SQLException{
    ContenedorDatos cd = new ContenedorDatos(this.metadatos.getContadorColumna());
    for (int i = 1;i <= this.metadatos.getContadorColumna();i++){
        cd.setNombreCampo(i, this.metadatos.getNombreCampo(i));
    }
    return cd;
}

public void setDataSource(DataSource dataSource){
    getDBServicios().setDataSource(dataSource);
}

public DataSource getDataSource() throws SQLException{
    return getDBServicios().getDataSource();
}

public void setDataSourceName(String dataSourceName){
    getDBServicios().setNombreOrigenDatos(dataSourceName);
}

public boolean isExceptionListaObjetosVacia() {
    return exceptionListaObjetosVacia;
}
```

```
    public void setExceptionListaObjetosVacia(boolean exceptionListaObjetosVacia) {
        this.exceptionListaObjetosVacia = exceptionListaObjetosVacia;
    }

    public boolean isCrearMetadatosDefault() {
        return crearMetadatosDefault;
    }

    public void setCrearMetadatosDefault(boolean crearMetadatosDefault) {
        this.crearMetadatosDefault = crearMetadatosDefault;
    }

    public boolean isThrowMissingFieldException() {
        return throwMissingFieldException;
    }

    public void setThrowMissingFieldException(boolean throwMissingFieldException) {
        this.throwMissingFieldException = throwMissingFieldException;
    }

    public Transformador getTransformador() {
        return transformador;
    }

    public void setTransformador(Transformador transformador) {
        this.transformador = transformador;
    }

    @Override
    public String toString() {
        StringBuffer sb = new StringBuffer();
        Object obj = null;
        Method[] methods = getClass().getMethods();
        for (int i = 0;i < methods.length;i++){
            String nombreMethod = methods[i].getName();
            if (nombreMethod.equals("getMetadatos") || nombreMethod.equals("getClass")){
                continue;
            }
            if(nombreMethod.equals("getDatosModificacionSequence") && methods[i].getParameterTypes().length == 0){
                if (datosModificacionSequence != null && datosModificacionSequence.length > 0){
                    sb.append("datosModificacionSequence=").append(new String(datosModificacionSequence)).append("&");
                }else{
                    sb.append("datosModificacionSequence=").append("&");
                }

            }else if ((nombreMethod.startsWith("get") || nombreMethod.startsWith("is")) && methods[i].getParameterTypes().length == 0){
                try{
                    obj = methods[i].invoke(this, EMPTY_READ_ARG);
                }catch (Throwable t){
                    continue;
                }
                String inicialCharCampo = "";
                if (nombreMethod.startsWith("is")){
                    inicialCharCampo = nombreMethod.substring(2, 3).toLowerCase();
                    sb.append(inicialCharCampo);
                    sb.append(nombreMethod.substring(3));
                }else if (nombreMethod.startsWith("get")){
                    inicialCharCampo = nombreMethod.substring(3, 4).toLowerCase();
                    sb.append(inicialCharCampo);
                    sb.append(nombreMethod.substring(4));
                }
                sb.append("=");
                sb.append((obj == null)?"":obj);
                sb.append("&");
            }
        }
        return sb.toString();
    }
}
```

Clase: GeneradorClaves.java

```
/*
   CREATE TABLE SEQUENCES(
      SEQUENCE_ID VARCHAR(30) NOT NULL,
      ID INTEGER NOT NULL,
      CONSTRAINT PK_SEQUENCE PRIMARY KEY(SEQUENCE_ID)
   );

   INSERT INTO SEQUENCES VALUES('MY_SEQUENCE', 0); // Para cada tabla...

   -HIGH/LOW ID modelo (máscara) de implementación-
 */

package org.fpuoc;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.math.BigInteger;
import java.math.BigDecimal;
import java.util.Collections;
import java.util.Set;
import java.util.HashSet;

public class GeneradorClaves {

   protected String tabla;    // obligatorio
   protected String columnaId; // obligatorio
   protected String sequenceNombreColumna; // obligatorio para distintas de externaSequence
   protected int incremento = 10; // cualquier numero superior a cero
   /** No debe usar la misma conexión para el resto de la aplicación sin realizar un commit siempre antes para GeneradorClaves!!! */
   protected DBServicios dbServicios;
   protected int maxRepeticiones = 3;   // cualquier numero superior a cero
   protected long timeout = 250; // cualquier numero superior a cero
   protected String estadoSQL = "23000"; // SQLSTATE - clave duplicada
   protected int codigoErrorVendedor = Integer.MIN_VALUE;
   protected int nivelAislamiento = - 1;
   protected boolean setNivelAislamiento = false;
   protected Map valoresBajos = Collections.synchronizedMap(new HashMap());
   protected Map valoresAltos = Collections.synchronizedMap(new HashMap());
   protected Class claveClass = Integer.class;   // cualquier clase que herede de Number
   protected boolean externaSequence = false; // se usará con ordenes (sequences) reales: Oracle, PostgreSQL, etc...
   protected String select;
   protected String update;
   protected String insert; // sin no existe se crea automaticamente una nueva orden (sequence) bajo petición
   protected boolean crearSequencePorSolicitud = true;
   protected Number valorInicial = new Integer(0); // no se usará con ordenes (sequences) reales
   protected Set sequences;
   protected boolean almacenStatements;


   public GeneradorClaves() {}

   public GeneradorClaves(int nivelAislamiento) {
      this.nivelAislamiento = nivelAislamiento;
   }

   public GeneradorClaves(String tabla, String sequenceNombreColumna, String columnaId, int nivelAislamiento) {
      this.tabla = tabla;
      this.sequenceNombreColumna = sequenceNombreColumna;
      this.columnaId = columnaId;
      this.nivelAislamiento = nivelAislamiento;
   }

   public GeneradorClaves(String tabla, String sequenceNombreColumna, String columnaId) {
      this.tabla = tabla;
      this.sequenceNombreColumna = sequenceNombreColumna;
      this.columnaId = columnaId;
   }

   public GeneradorClaves(String tabla, String sequenceNombreColumna, String columnaId, int nivelAislamiento, boolean externaSequence) {
```

```
        this.tabla = tabla;
        this.sequenceNombreColumna = sequenceNombreColumna;
        this.columnaId = columnaId;
        this.nivelAislamiento = nivelAislamiento;
        this.externaSequence = externaSequence;
    }

    public GeneradorClaves(String tabla, String sequenceNombreColumna, String columnaId, boolean externaSequence) {
        this.tabla = tabla;
        this.sequenceNombreColumna = sequenceNombreColumna;
        this.columnaId = columnaId;
        this.externaSequence = externaSequence;
    }

    public void init(){
        if (!externaSequence && this.tabla == null){
            throw new IllegalStateException("Nombre tabla no existe");
        }
        if (!externaSequence && this.sequenceNombreColumna == null){
            throw new IllegalStateException("Nombre columna de orden (sequence) no existe");
        }
        if (this.columnaId == null){
            throw new IllegalStateException("ID de nombre de columna no existe");
        }
        if (dbServicios == null){
            throw new IllegalStateException("DBServicios no establecido");
        }
        if (!externaSequence){
            // SELECT ID FROM SEQUENCES WHERE SEQUENCE_NAME = ? ${sequence}
            select = "SELECT " + columnaId + " FROM " + tabla + " WHERE " + sequenceNombreColumna + " = ?";
            // UPDATE SEQUENCES SET ID = ID + 10 WHERE SEQUENCE_NAME = ? AND ID = ? ${current ID}
            update = "UPDATE " + tabla + " SET " + columnaId + " = " + columnaId + " + " + String.valueOf(incremento) + " WHERE " +
sequenceNombreColumna + " = ? AND " + columnaId + " = ?";
            if (crearSequencePorSolicitud){
                // INSERT INTO SEQUENCES (SEQUENCE_NAME, ID) VALUES (?, 0) ${sequence}
                insert = "INSERT INTO " + tabla + " (" + sequenceNombreColumna + ", " + columnaId + ") VALUES (?, " + valorInicial.toString() + ")";
            }
            String totalSequences = "SELECT " + sequenceNombreColumna + " FROM " + tabla;
            try{
                sequences = Collections.synchronizedSet(new HashSet(dbServicios.getResultadoUnicaColumnaComoLista(totalSequences)));
            }catch (SQLException sqle){
                throw new RuntimeException(sqle);
            }
        }
    }

    public synchronized Number getValorSiguiente(String sequence) throws SQLException{
        if (!externaSequence && this.select == null){
            throw new IllegalStateException("GeneradorClaves no inicializado");
        }
        if (!externaSequence && !sequences.contains(sequence) && crearSequencePorSolicitud){ // nueva orden (sequence)
            Collection colSeq = new ArrayList(1);
            colSeq.add(sequence);
            dbServicios.ejecutar(insert, colSeq);
            sequences.add(sequence);
        }
        Object valor = valoresBajos.get(sequenceNombreColumna);
        Number valorAlto = null;
        if (valor == null){
            valor = getSemilla(sequence);
            valoresBajos.put(sequenceNombreColumna, valor);
            if (claveClass.getName().equals("Integer")){
                valorAlto = ((Number) valor).intValue() + incremento;
            }else if (claveClass.getName().equals("Long")){
                valorAlto = ((Number) valor).longValue() + incremento;
            }else if (claveClass.getName().equals("java.math.BigInteger")){
                valorAlto = new BigInteger(valor.toString()).add(new BigInteger(String.valueOf(incremento)));
            }else if (claveClass.getName().equals("java.math.BigDecimal")){
                valorAlto = new BigDecimal(valor.toString()).add(new BigDecimal(String.valueOf(incremento)));
            }else{
                try{
                    valorAlto = new BigDecimal(valor.toString()).add(new BigDecimal(String.valueOf(incremento)));
                    Constructor c = claveClass.getConstructor(String.class);
                    valorAlto = (Number) c.newInstance(valorAlto.toString());
                }catch (InvocationTargetException ite){
                    throw new SQLException(ite.getTargetException().toString());
                }catch (Exception e){
                    throw new SQLException(e.toString());
                }
            }
            valoresAltos.put(sequenceNombreColumna, valorAlto);
```

```
    }else{
      if (claveClass.getName().equals("Integer")){
        if (((Number) valor).intValue() < ((Number) valoresAltos.get(sequenceNombreColumna)).intValue() - 1){
          valor = new Integer(((Number) valor).intValue() + 1);
          valoresBajos.put(sequenceNombreColumna, valor);
        }else{
          valor = getSemilla(sequence);
          valoresBajos.put(sequenceNombreColumna, valor);
          valoresAltos.put(sequenceNombreColumna, ((Number) valor).intValue() + incremento);
        }
      }else if (claveClass.getName().equals("Long")){
        if (((Number) valor).longValue() < ((Number) valoresAltos.get(sequenceNombreColumna)).longValue() - 1){
          valor = new Long(((Number) valor).longValue() + 1);
          valoresBajos.put(sequenceNombreColumna, valor);
        }else{
          valor = getSemilla(sequence);
          valoresBajos.put(sequenceNombreColumna, valor);
          valoresAltos.put(sequenceNombreColumna, ((Number) valor).longValue() + incremento);
        }
      }else if (claveClass.getName().equals("java.math.BigInteger")){
        if (new BigInteger(valor.toString()).compareTo(new BigInteger(valoresAltos.get(sequenceNombreColumna).toString()))  == -1){
          valor = new BigInteger(valor.toString()).add(BigInteger.ONE);
          valoresBajos.put(sequenceNombreColumna, valor);
        }else{
          valor = getSemilla(sequence);
          valoresBajos.put(sequenceNombreColumna, valor);
          valoresAltos.put(sequenceNombreColumna, new BigInteger(valor.toString()).add(new BigInteger(String.valueOf(incremento))));
        }
      }else if (claveClass.getName().equals("java.math.BigDecimal")){
        if (new BigDecimal(valor.toString()).compareTo(new BigDecimal(valoresAltos.get(sequenceNombreColumna).toString()))  == -1){
          valor = new BigDecimal(valor.toString()).add(BigDecimal.ONE);
          valoresBajos.put(sequenceNombreColumna, valor);
        }else{
          valor = getSemilla(sequence);
          valoresBajos.put(sequenceNombreColumna, valor);
          valoresAltos.put(sequenceNombreColumna, new BigDecimal(valor.toString()).add(new BigDecimal(String.valueOf(incremento))));
        }
      }else{
        if (new BigDecimal(valor.toString()).compareTo(new BigDecimal(valoresAltos.get(sequenceNombreColumna).toString()))  == -1){
          valor = new BigDecimal(valor.toString()).add(BigDecimal.ONE);
          try{
            valor = new BigDecimal(valor.toString()).add(BigDecimal.ONE);
            Constructor c = claveClass.getConstructor(String.class);
            valoresBajos.put(sequenceNombreColumna, c.newInstance(valor.toString()));
          }catch (InvocationTargetException ite){
            throw new SQLException(ite.getTargetException().toString());
          }catch (Exception e){
            throw new SQLException(e.toString());
          }
        }else{
          valor = getSemilla(sequence);
          valoresBajos.put(sequenceNombreColumna, valor);
          try{
            Constructor c = claveClass.getConstructor(String.class);
            BigDecimal temp = new BigDecimal(valor.toString()).add(new BigDecimal(String.valueOf(incremento)));
            valoresAltos.put(sequenceNombreColumna, c.newInstance(temp.toString()));
          }catch (InvocationTargetException ite){
            throw new SQLException(ite.getTargetException().toString());
          }catch (Exception e){
            throw new SQLException(e.toString());
          }
        }
      }
    }
    return (Number) valor;
  }

  public synchronized Number getValorActual(String sequence) throws SQLException{
    if (!externaSequence && this.select == null){
      throw new IllegalStateException("GeneradorClaves no inicializado");
    }
    if (!externaSequence && !sequences.contains(sequence) && crearSequencePorSolicitud){ // nueva orden (sequence)
      Collection colSeq = new ArrayList(1);
      colSeq.add(sequence);
      dbServicios.ejecutar(insert, colSeq);
      sequences.add(sequence);
    }
    Object valorActual;
    try{
```

```
        if (externaSequence){
          if (tabla == null || tabla.trim().length() == 0){
            valorActual = dbServicios.getValorUnico("SELECT " + columnaId);
          }else{
            valorActual = dbServicios.getValorUnico("SELECT " + columnaId + " FROM " + tabla);
          }
        }else{
          Collection colSeq = new ArrayList(1);
          colSeq.add(sequence);
          valorActual = dbServicios.getValorUnico(select, colSeq);
        }
      }catch(SQLException sqle){
        throw sqle;
      }finally{
        dbServicios.release(false);
      }
      return (Number) valorActual;
    }

    protected synchronized Number getSemilla(String sequence) throws SQLException{
      if (!externaSequence && this.select == null){
        throw new IllegalStateException("GeneradorClaves no inicializado");
      }
      Object id = null;
      boolean conseguido = false;
      if (externaSequence){
        try{
          id = dbServicios.getValorUnico("SELECT " + columnaId + " FROM " + tabla);
          conseguido = true;
        }finally{
          dbServicios.release(false);
        }
      }else{
        try{
          int contadorRepeticiones = 0;
          if (nivelAislamiento != -1 && !setNivelAislamiento){ // colocarlo solo una vez
            dbServicios.getConnection().setTransactionIsolation(nivelAislamiento);
            setNivelAislamiento = true;
          }
          dbServicios.beginTran();
          while (!conseguido && contadorRepeticiones <= maxRepeticiones){
            try{
              Collection colSeq = new ArrayList(1);
              colSeq.add(sequence);
              System.out.println("========== colSeq: " + colSeq);
              id = dbServicios.getValorUnico(select, colSeq);
              colSeq.add(id);
              dbServicios.ejecutar(update, colSeq);
              conseguido = true;
              if (claveClass.getName().equals("Integer")){
                id = new Integer(((Number) id).intValue() + incremento);
              }else if (claveClass.getName().equals("Long")){
                id = new Long(((Number) id).intValue() + incremento);
              }else if (claveClass.getName().equals("java.math.BigInteger")){
                id = new BigInteger(id.toString()).add(new BigInteger(String.valueOf(incremento)));
              }else if (claveClass.getName().equals("java.math.BigDecimal")){
                id = new BigDecimal(id.toString()).add(new BigDecimal(String.valueOf(incremento)));
              }else{
                Constructor c = claveClass.getConstructor(String.class);
                id = c.newInstance(String.valueOf(incremento));
              }
              break;
            } catch (InstantiationException ie) {
              throw new SQLException(ie.toString());
            } catch (IllegalAccessException iae) {
              throw new SQLException(iae.toString());
            } catch (InvocationTargetException ite) {
              throw new SQLException(ite.getTargetException().toString());
            }catch(SQLException sqle){
              if ((sqle.getSQLState() != null && sqle.getSQLState().equals(estadoSQL)) || sqle.getErrorCode() == codigoErrorVendedor){
                // clave duplicada: reintentar
              }else{
                throw sqle; // cualquier otro problema
              }
            }
            contadorRepeticiones++;
            try{
              wait(timeout);
            }catch(InterruptedException ie){
              throw new SQLException("No puede recibir ID único");
            }
```

```
            }
        }catch(NoSuchMethodException nsme){
            throw new SQLException(nsme.getMessage()); // no debería suceder
        }catch(SQLException sqle){
            throw sqle;
        }finally{
            dbServicios.commitTran(); // commit debe realizarse en cualquier caso
            dbServicios.release(false);
        }
    }
    if (!conseguido){
        throw new SQLException("No puede recibir ID único");
    }
    return (Number) id;
}

public String getTabla() {
    return tabla;
}

public void setTabla(String tabla) {
    this.tabla = tabla;
}

public String getColumnaId() {
    return columnaId;
}

public void setColumnaId(String columnaId) {
    this.columnaId = columnaId;
}

public int getIncremento() {
    return incremento;
}

public void setIncremento(int incremento) {
    this.incremento = incremento;
}

public DBServicios getDBServicios() {
    return dbServicios;
}

public void setDBServicios(DBServicios dbServicios) {
    this.dbServicios = dbServicios;
    this.dbServicios.setAlmacenStatements(almacenStatements);
}

public int getMaxRepeticiones() {
    return maxRepeticiones;
}

public void setMaxRepeticiones(int maxRepeticiones) {
    this.maxRepeticiones = maxRepeticiones;
}

public String getEstadoSQL() {
    return estadoSQL;
}

public void setEstadoSQL(String estadoSQL) {
    this.estadoSQL = estadoSQL;
}

public long getTimeout() {
    return timeout;
}

public void setTimeout(long timeout) {
    if (timeout < 0){
        throw new IllegalArgumentException("Timeout debe ser superior a cero");
    }
    this.timeout = timeout;
}

public int getNivelAislamiento() {
    return nivelAislamiento;
}
```

```
public void setNivelAislamiento(int nivelAislamiento) {
   this.nivelAislamiento = nivelAislamiento;
}

public String getSequenceNombreColumna() {
   return sequenceNombreColumna;
}

public void setSequenceNombreColumna(String sequenceNombreColumna) {
   this.sequenceNombreColumna = sequenceNombreColumna;
}

public Class getClaveClass() {
   return claveClass;
}

public void setClaveClass(Class claveClass) {
   if(Number.class.isAssignableFrom(claveClass)){
      this.claveClass = claveClass;
   }else{
      throw new IllegalArgumentException("claveClass debe heredar de Number");
   }
   try{
      Constructor c = claveClass.getConstructor(String.class);
      Object obj = c.newInstance("0");
      String valorString = obj.toString();
      if (valorString == null && !(valorString.equals("0") && !(valorString.startsWith("0.")))){
         throw new IllegalArgumentException(claveClass.getName() + ".toString() debe devolver un valor String o Number");
      }
   }catch (InvocationTargetException ite){
      throw new IllegalArgumentException(ite.getTargetException().toString());
   }catch (Exception ex){
      throw new IllegalArgumentException(ex.toString());
   }
}

public boolean isExternaSequence() {
   return externaSequence;
}

public void setExternaSequence(boolean externaSequence) {
   this.externaSequence = externaSequence;
}

public Number getValorInicial() {
   return valorInicial;
}

public void setValorInicial(Number valorInicial) {
   this.valorInicial = valorInicial;
}

public boolean isAlmacenStatements() {
   return almacenStatements;
}

public void setAlmacenStatements(boolean almacenStatements) {
   this.almacenStatements = almacenStatements;
}

public boolean isCrearSequencePorSolicitud() {
   return crearSequencePorSolicitud;
}

public void setCrearSequencePorSolicitud(boolean crearSequencePorSolicitud) {
   this.crearSequencePorSolicitud = crearSequencePorSolicitud;
}

@SuppressWarnings("empty-statement")
public void reset() throws SQLException{
   tabla = null;
   columnaId = null;
   sequenceNombreColumna = null;
   incremento = 10;
   maxRepeticiones = 3;
   timeout = 250;
   codigoErrorVendedor = Integer.MIN_VALUE;
   nivelAislamiento = - 1;
   setNivelAislamiento = false;
   valoresBajos.clear();
```

```
        valoresAltos.clear();
        claveClass = Integer.class;
        externaSequence = false;
        select = null;
        update = null;;
        insert = null;
        valorInicial = new Integer(0);
        almacenStatements = false;
        sequences.clear();
        dbServicios.release(true);
    }

}
```

Clase: GrabarDatos.java

```
package org.fpuoc;

import java.util.List;

public interface GrabarDatos extends java.io.Serializable, Cloneable {
    public boolean grabaDatos(FactoriaObjetoPersistente peticion);
    public void reset();
    public int getContadorModificaciones();
    public int getContadorAltas();
    public int getContadorBajas();
    public List getSQLGeneradas();
    public String getGenerarSentenciasSQLcomoXML();
    public Throwable getErroresDetectados();
    public boolean isExcepcionEnObjectListVacios();
    public void setExcepcionEnObjectListVacios(boolean excepcionEnObjectListVacios);
}
```

Clase: ManejadorExcepciones.java

```
package org.fpuoc;

import java.util.Collection;

public interface ManejadorExcepciones extends java.io.Serializable{
    /** Manejador de excepciones en el código
     *  Los métodos deben devolver 'true' para continuar y 'false' para lanzar
     *  la excepción mediante Throwable t
     */
    public boolean manejadorExcepcion (ObjetoPersistente objetoPersistente, String sql, Collection colParametros, Throwable t);
    public boolean manejadorExcepcion (ObjetoPersistente objetoPersistente, String sql, Throwable t);
    public boolean manejadorExcepcion (ObjetoPersistente objetoPersistente, Throwable t);
    public boolean manejadorExcepcion (ObjetoPersistente objetoPersistente, String campo, Object valor, Throwable t);
    public void setFactoriaObjetoPersistente(FactoriaObjetoPersistente manejadorFactoria);
}
```

Clase: MarcadorPagina.java

```
package org.fpuoc;

import java.util.List;

public class MarcadorPagina implements Cloneable, java.io.Serializable {

   protected int totalPaginas = 0;
   protected int numeroPagina = 0;
   protected int objetosTotales = 0;
   protected List pagina = null;

   /** Crea una nueva instancia de MarcadorPagina */
   public MarcadorPagina() {
   }

   public MarcadorPagina(int numeroPagina, int totalPaginas, int objetosTotales, List pagina) {
      this.numeroPagina = numeroPagina;
      this.totalPaginas = totalPaginas;
      this.objetosTotales = objetosTotales;
      this.pagina = pagina;
   }

   /** Leer la propiedad de totalPaginas.
    * @return Value of property totalPaginas.
    *
    */
   public int getTotalPaginas() {
      return totalPaginas;
   }

   /** Asignar nueva propiedad a totalPaginas.
    * @param totalPaginas New value of property totalPaginas.
    *
    */
   public void setTotalPaginas(int totalPaginas) {
      this.totalPaginas = totalPaginas;
   }

   /** Leer la propiedad de numeroPagina.
    * @return Value of property numeroPagina.
    *
    */
   public int getNumeroPagina() {
      return numeroPagina;
   }

   /** Asignar nueva propiedad a numeroPagina.
    * @param numeroPagina New value of property numeroPagina.
    *
    */
   public void setNumeroPagina(int numeroPagina) {
      this.numeroPagina = numeroPagina;
   }

   /** Leer la propiedad de pagina.
    * @return Value of property pagina.
    *
    */
   public List getPagina() {
      return pagina;
   }

   /** Asignar nueva propiedad a pagina.
    * @param pagina New value of property pagina.
    *
    */
   public void setPagina(List pagina) {
      this.pagina = pagina;
   }

   @Override
   public String toString(){
      StringBuffer sb = new StringBuffer();
      sb.append("numeroPagina=").append(numeroPagina);
      sb.append("&totalPaginas=").append(totalPaginas);
      sb.append("&objetosTotales=").append(objetosTotales);
```

```
        sb.append("&pagina=").append(pagina);
        return sb.toString();
    }

    /** Leer la propiedad de objetosTotales.
     * @return Value of property objetosTotales.
     *
     */
    public int getObjetosTotales() {
        return objetosTotales;
    }

    /** Asignar nueva propiedad a objetosTotales.
     * @param objetosTotales New value of property objetosTotales.
     *
     */
    public void setObjetosTotales(int objetosTotales) {
        this.objetosTotales = objetosTotales;
    }

}
```

Clase: Metadatos.java

```
package org.fpuoc;

import javax.sql.*;
import java.util.*;
import java.sql.SQLException;
import java.io.Serializable;
import java.lang.reflect.*;
import java.beans.*;

public class Metadatos implements Serializable, Cloneable {

    protected boolean[] autoIncremento = null;
    protected boolean[] distingueMayMin = null;
    protected boolean[] moneda = null;
    protected boolean[] finalmenteGrabable = null;
    protected boolean[] soloLectura = null;
    protected boolean[] localizable = null;
    protected boolean[] firmado = null;
    protected boolean[] grabable = null;
    protected int[] columnaDisplaySize = null;
    protected int[] tipoColumna = null;
    protected int[] tipoNull = null;
    protected int[] precision = null;
    protected int[] escala = null;
    protected String[] nombreCatalogo = null;
    protected String[] nombreEsquema = null;
    protected String[] nombreTabla = null;
    protected String[] columnaNombreClass = null;
    protected String[] etiquetaColumna = null;
    protected String[] nombreColumna = null;
    protected String[] columnaNombreTipo = null;
    protected String[] completoNombreColumna = null;
    protected String[] nombreCampo = null;
    /**
     * reflect.Method no es serializable, por lo que despues de enviar
     * este objeto a otra maquina virtual Java (JavaVM) grabarMethod y
     * leerMethod serán de tipo null
     */
    protected transient Method[] leerMethod = null;
    protected transient Method[] grabarMethod = null;
    protected String[] campoNombreClass = null;
    protected Class[] campoClass = null;
    protected int contadorColumna = 0;
    protected Class solicitante = null; // ObjetoPersistente

    protected Map tablasMap = new HashMap();
    // Lanza o no una Excepcion considerando si ObjetoPersistente tiene menos campos que mapeos
    protected boolean throwExcepcionCampoAusente = false;

    /**
```

```
 * Almacena y habilita información sobre campos, tablas, columnas, etc.
 *
 */
public Metadatos() {
}

/** Lee el nombre de catálogo de la tabla de la columna designada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return nombre del catálogo para la tabla en la que la columna
 *         en particular aparece o "" si no es aplicable
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getNombreCatalogo(int columnaIdx) throws SQLException {
   return this.nombreCatalogo[columnaIdx - 1];
}

/**
 * <p>Devuelve el nombre completo calificado de la clase Java cuyas instancias
 * han sido generadas si el método <code>ResultSet.getObject</code>
 * se le llama para recuperar un valor de la columna.
 * <code>ResultSet.getObject</code> puede devolver una subclase de la clase
 * devuelta por este método.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return el nombre calificado completo de la clase en el lenguaje Java
 *         que debe ser usada por este método
 * <code>ResultSet.getObject</code> para recuperar el valor en la columna
 * especificada. Este es el nombre d ela clase para el mapeo directo.
 * @exception SQLException si sucede un error de acceso a BBDD
 * @since 1.2
 *
 */
public String getColumnaNombreClass(int columnaIdx) throws SQLException {
   return this.columnaNombreClass[columnaIdx - 1];
}

/** Devuelve el numero de columnas en este objeto <code>ResultSet</code>.
 *
 * @return numero de columnas
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public int getContadorColumna() throws SQLException {
   return this.contadorColumna;
}

/** Indica el ancho máximo nommal (en caracteres) para la columna designada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return el número maximo normal permitido de caracteres como ancho
 *         de la columna designada
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public int getColumnaDisplaySize(int columnaIdx) throws SQLException {
   return this.columnaDisplaySize[columnaIdx - 1];
}

/** Lee el título indicado de la columna designada para su uso en
 * impresiones y visualizaciones.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return título de columna indicado
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getEtiquetaColumna(int columnaIdx) throws SQLException {
   return this.etiquetaColumna[columnaIdx - 1];
}

/** Lee el nombre la columna seleccionada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return nombre columna
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getNombreColumna(int columnaIdx) throws SQLException {
   return this.nombreColumna[columnaIdx - 1];
```

```
}

/** Recupera el tipo de SQL de la columna designada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return tipo SQL en formato de java.sql.Types
 * @exception SQLException si sucede un error de acceso a BBDD
 * @see java.sql.Types
 *
 */
public int getTipoColumna(int columnaIdx) throws SQLException {
   return this.tipoColumna[columnaIdx - 1];
}

/** Recupera el tipo de nombre específico de la BBDd para
 * la columna seleccionada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return nombre de tipo usado por la BBDD. Si el tipo de columna es
 * un tipo definido por el usuario, entonces se devuelve un nombre de tipo
 * completamente calificado.
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getColumnaNombreTipo(int columnaIdx) throws SQLException {
   return this.columnaNombreTipo[columnaIdx - 1];
}

/** Devuelve el número de digitos decimales de la columna indicada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return precision
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public int getPrecision(int columnaIdx) throws SQLException {
   return this.precision[columnaIdx - 1];
}

/** Devuelve el número de digitos a la derecha del punto decimal
 * en la columna indicada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return escala
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public int getEscala(int columnaIdx) throws SQLException {
   return this.escala[columnaIdx - 1];
}

/** Devuelve el esquema de la tabla de la columna designada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return nombre del esquema o "" si no es posible
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getNombreEsquema(int columnaIdx) throws SQLException {
   return this.nombreEsquema[columnaIdx - 1];
}

/** Devuelve el nombre de tabla de la columna designada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return nombre de la tabla o "" si no es posible
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public String getNombreTabla(int columnaIdx) throws SQLException {
   return this.nombreTabla[columnaIdx - 1];
}

/** Indica si la columna designada se autonumera automáticamente, siendo
 * de solo lectura
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
```

```
 *
 */
public boolean isAutoIncremento(int columnaIdx) throws SQLException {
   return this.autoIncremento[columnaIdx - 1];
}

/** Indica si distingue entre Mayúsculas y minúsculas
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isDistingueMayMin(int columnaIdx) throws SQLException {
   return this.distingueMayMin[columnaIdx - 1];
}

/** Indica si la columna es de un valor monetario.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isMoneda(int columnaIdx) throws SQLException {
   return this.moneda[columnaIdx - 1];
}

/** Indica que la escritura en dicha columna es definitiva.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isFinalmenteGrabable(int columnaIdx) throws SQLException {
   return this.finalmenteGrabable[columnaIdx - 1];
}

/** Indica que los valores de la columna son anulables (null).
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return el estado de anulabilidad de la columna;
 *         uno de los siguientes: <code>columnNoNulls</code>,
 *         <code>columnNullable</code> o <code>columnNullableUnknown</code>
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public int isTipoNull(int columnaIdx) throws SQLException {
   return this.tipoNull[columnaIdx - 1];
}

/** Indica que la columna designada no será grabada.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isSoloLectura(int columnaIdx) throws SQLException {
   return this.soloLectura[columnaIdx - 1];
}

/** indica que la columna puede ser usada en una cláusula WHERE
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isLocalizable(int columnaIdx) throws SQLException {
   return this.localizable[columnaIdx - 1];
}

/** Indica si los valores en la columna designada son los números firmados.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
```

```
public boolean isFirmado(int columnaIdx) throws SQLException {
   return this.firmado[columnaIdx - 1];
}

/** Indica si es posible que una escritura sobre la columna designada dé resultado.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @return <code>true</code> si lo es; <code>false</code> de otra forma
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public boolean isGrabable(int columnaIdx) throws SQLException {
   return this.grabable[columnaIdx - 1];
}

/**
 * Asigna si la columna designada será automáticamente numerada,
 * y de solo lectura. Por defecto las columnas de un objeto <code>RowSet</code>
 * no son automáticamente numeradas.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param propiedad <code>true</code> si la columna se autonumera
 *            <code>false</code> en otro caso
 *
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setAutoIncremento(int columnaIdx, boolean propiedad) throws SQLException {
   this.autoIncremento[columnaIdx - 1] = propiedad;
}

/** Asigna la columna para distinguir mayúsculas de minúsculas.
 * Por defecto es <code>false</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param propiedad <code>true</code> si la columna las distingue;
 *            <code>false</code> en otro caso
 *
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setDistingueMayMin(int columnaIdx, boolean propiedad) throws SQLException {
   this.distingueMayMin[columnaIdx - 1] = propiedad;
}

/** Asigna, si existe, el nombre de catálogo a la columna designada, en el
 * <code>String</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param nombreCatalogo the column's catalogo name
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setNombreCatalogo(int columnaIdx, String nombreCatalogo) throws SQLException {
   this.nombreCatalogo[columnaIdx - 1] = nombreCatalogo;
}

/** Asigna el numero de columnas en el objeto <code>RowSet</code> en el
 * valor númerico.
 *
 * @param contadorColumna the number of columns in the <code>RowSet</code> object
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setContadorColumna(int contadorColumna) throws SQLException {
   this.contadorColumna = contadorColumna;
}

/** Asigna el ancho máximo normal de la columna (en caracteres) en el
 * valor <code>int</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param longitud en caracteres máxima de ancho de columna
 *
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setColumnaDisplaySize(int columnaIdx, int longitud) throws SQLException {
   this.columnaDisplaySize[columnaIdx - 1] = longitud;
```

```
}

/** Asigna el titulo de columna indicado, si lo hay, para usarlo en impresiones
 * o en visualizaciones en el <code>String</code>
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param etiqueta con el titulo de la columna
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setEtiquetaColumna(int columnaIdx, String etiqueta) throws SQLException {
   this.etiquetaColumna[columnaIdx - 1] = etiqueta;
}

/** Asigna el nombre de la colunmna designada al indicado <code>String</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param nombreColumna el nombre de columna designado
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setNombreColumna(int columnaIdx, String nombreColumna) throws SQLException {
   this.nombreColumna[columnaIdx - 1] = nombreColumna;
}

/** Asigna el tipo SQL de la columna de uno de los indicados.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param tipoSQL el tipo SQL de la columna
 * @exception SQLException si sucede un error de acceso a BBDD
 * @see java.sql.Types
 *
 */
public void setTipoColumna(int columnaIdx, int tipoSQL) throws SQLException {
   this.tipoColumna[columnaIdx - 1] = tipoSQL;
}

/** Asigna el nombre de tipo de la columna designada que es especifico
 * del origen de datos, si lo hay, al <code>String</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param nombreTipo nombre de tipo especifico de la fuente de datos.
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setColumnaNombreTipo(int columnaIdx, String nombreTipo) throws SQLException {
   this.columnaNombreTipo[columnaIdx - 1] = nombreTipo;
}

/** Asigna la columna como para valores monetarios.
 * Por defecto es <code>false</code>.
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param propiedad <code>true</code> si la columna contiene un valor monetario;
 *              <code>false</code> en otro caso
 *
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setMoneda(int columnaIdx, boolean propiedad) throws SQLException {
   this.moneda[columnaIdx - 1] = propiedad;
}

/** Asigna que la columna indicada puede ser puesta con el valor
 * <code>NULL</code>.
 * Por defecto es <code>ResultSetMetaData.columnNullableUnknown</code>
 *
 * @param columnaIdx la primera columna es 1, la segunda 2, ...
 * @param propiedad una de las constantes siguientes:
 *              <code>ResultSetMetaData.columnNoNulls</code>,
 *              <code>ResultSetMetaData.columnNullable</code>, o
 *              <code>ResultSetMetaData.columnNullableUnknown</code>
 *
 * @exception SQLException si sucede un error de acceso a BBDD
 *
 */
public void setTipoNull(int columnaIdx, int propiedad) throws SQLException {
   this.tipoNull[columnaIdx - 1] = propiedad;
}

/** Asigna el número de digitos decimales de la columna en el valor
```

```
     * indicado <code>int</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param precision el numero total de digitos decimales
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setPrecision(int columnaIdx, int precision) throws SQLException {
        this.precision[columnaIdx - 1] = precision;
    }

    /** Asigna el número de digitos a la derecha del punto decimal para la
     * columna indicada en el valor <code>int</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param escala numeor de digitos a la derecha del punto decimal
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setEscala(int columnaIdx, int escala) throws SQLException {
        this.escala[columnaIdx - 1] = escala;
    }

    /** Asigna el nombre del esquema de la tabla de la columna designada, si lo hay,
     * en el indicado <code>String</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param nombreEsquema nombre de esquema
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setNombreEsquema(int columnaIdx, String nombreEsquema) throws SQLException {
        this.nombreEsquema[columnaIdx - 1] = nombreEsquema;
    }

    /** Asigna a la columna para poder usarse en las clausulas WHEREe.
     * Por defecto es <code>false</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param propiedad <code>true</code> si la columna puede usarse en una
     *              clausula <code>WHERE</code>; <code>false</code> en otro caso
     *
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setLocalizable(int columnaIdx, boolean propiedad) throws SQLException {
        this.localizable[columnaIdx - 1] = propiedad;
    }

    /** Asigna a la columna como contenedora de numeros firmados (signed).
     * Por defecto es <code>false</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param propiedad <code>true</code> si la columna contiene un numero 'signed';
     *              <code>false</code> en otro caso
     *
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setFirmado(int columnaIdx, boolean propiedad) throws SQLException {
        this.firmado[columnaIdx - 1] = propiedad;
    }

    /** Asigna el nombre de tabla de la columna, si lo hay, en el valor
     * <code>String</code>.
     *
     * @param columnaIdx la primera columna es 1, la segunda 2, ...
     * @param nombreTabla nombre de tabla de la columna
     * @exception SQLException si sucede un error de acceso a BBDD
     *
     */
    public void setNombreTabla(int columnaIdx, String nombreTabla) throws SQLException {
        this.nombreTabla[columnaIdx - 1] = nombreTabla;
    }

    public void setGrabable(int columnaIdx, boolean grabable) {
        this.grabable[columnaIdx - 1] = grabable;
        this.soloLectura[columnaIdx - 1] = !grabable;
    }
```

```
public void setFinalmenteGrabable(int columnIndex, boolean finalmenteGrabable) {
   this.finalmenteGrabable[columnIndex - 1] = finalmenteGrabable;
}

public void setColumnaNombreClass(int columnaIdx, String columnaNombreClass) {
   this.columnaNombreClass[columnaIdx - 1] = columnaNombreClass;
}

public void setSoloLectura(int columnaIdx, boolean soloLectura) throws SQLException {
   this.soloLectura[columnaIdx - 1] = soloLectura;
   this.grabable[columnaIdx - 1] = !soloLectura;
}

public Map getTablasMap() {
   return tablasMap;
}

public void setTablasMap(Map tablasMap) {
   this.tablasMap = tablasMap;
}

public String getCompletoNombreColumna(int columnaIdx) throws SQLException {
   return this.completoNombreColumna[columnaIdx - 1];
}

public void setCompletoNombreColumna(int columnaIdx, String nombreColumna) throws SQLException {
   this.completoNombreColumna[columnaIdx - 1] = nombreColumna;
}

public String getNombreCampo(int columnIndex) throws SQLException {
   return this.nombreCampo[columnIndex - 1];
}

public void setNombreCampo(int columnaIdx, String nombreCampo) throws SQLException {
   this.nombreCampo[columnaIdx - 1] = nombreCampo;
}

public Method getGrabarMethod(int columnaIdx) throws SQLException {
   if (this.grabarMethod == null || this.grabarMethod[columnaIdx - 1] == null) {
      llenarGrabarMethod(columnaIdx, nombreCampo[columnaIdx - 1]);
   }
   return this.grabarMethod[columnaIdx - 1];
}

public void setGrabarMethod(int columnaIdx, Method grabarMethod) {
   if (this.grabarMethod == null) {
      this.grabarMethod = new Method[contadorColumna];
   }
   this.grabarMethod[columnaIdx - 1] = grabarMethod;
}

public Method getLeerMethod(int columnaIdx) throws SQLException {
   if (this.leerMethod == null || this.leerMethod[columnaIdx - 1] == null) {
      llenarLeerMethod(columnaIdx, nombreCampo[columnaIdx - 1]);
   }
   return this.leerMethod[columnaIdx - 1];
}

public void setLeerMethod(int columnaIdx, Method leerMethod) {
   if (this.leerMethod == null) {
      this.leerMethod = new Method[contadorColumna];
   }
   this.leerMethod[columnaIdx - 1] = leerMethod;
}

public String getCampoNombreClass(int columnaIdx) throws SQLException {
   return this.campoNombreClass[columnaIdx - 1];
}

public void setCampoNombreClass(int columnaIdx, String campoNombreClass) throws SQLException {
   this.campoNombreClass[columnaIdx - 1] = campoNombreClass;
}

public Class getCampoClass(int columnaIdx) {
   return this.campoClass[columnaIdx - 1];
}

public void setCampoClass(int columnaIdx, Class campoClass) {
   this.campoClass[columnaIdx - 1] = campoClass;
}
```

```
protected void llenarLeerMethod(int columnaIdx, String nombreCampo) throws java.sql.SQLException {
   try {
      Method lMethod = (new PropertyDescriptor(nombreCampo, this.solicitante)).getReadMethod();
      this.setLeerMethod(columnaIdx, lMethod);
   } catch (IntrospectionException ie) {
      if (throwExcepcionCampoAusente) {
         ie.printStackTrace();
         throw new RuntimeException(ie);
      }
   }
}

protected void llenarGrabarMethod(int columnaIdx, String nombreCampo) throws java.sql.SQLException {
   try {
      Method gMethod = (new PropertyDescriptor(nombreCampo, this.solicitante)).getWriteMethod();
      this.setGrabarMethod(columnaIdx, gMethod);
      Class[] tiposParms = gMethod.getParameterTypes();
      Class paramClass = tiposParms[0];
      this.setCampoClass(columnaIdx, paramClass);
      this.setCampoNombreClass(columnaIdx, paramClass.getName());
   } catch (IntrospectionException ie) {
      if (throwExcepcionCampoAusente) {
         ie.printStackTrace();
         throw new RuntimeException(ie);
      }
   }
}

public void llenar(Map mapping, Class solicitante) throws java.sql.SQLException {
   if (mapping == null || mapping.isEmpty()) {
      throw new SQLException("Metadatos: invalido/ausente mapping");
   }
   this.solicitante = solicitante;
   int contColumna = mapping.size();
   this.setContadorColumna(contColumna);
   this.nombreCatalogo = new String[contColumna];
   this.nombreEsquema = new String[contColumna];
   this.nombreTabla = new String[contColumna];
   this.nombreColumna = new String[contColumna];
   this.etiquetaColumna = new String[contColumna];
   this.soloLectura = new boolean[contColumna];
   this.grabable = new boolean[contColumna];
   this.tipoColumna = new int[contColumna];
   this.completoNombreColumna = new String[contColumna];
   this.nombreCampo = new String[contColumna];
   this.leerMethod = new Method[contColumna];
   this.grabarMethod = new Method[contColumna];
   this.campoClass = new Class[contColumna];
   this.campoNombreClass = new String[contColumna];

   Iterator iter = mapping.keySet().iterator();
   int columnaIdx = 1;
   while (iter.hasNext()) {
      String catalogo = null;
      String esquema = null;
      String tabla = null;
      String tablaId = null;
      String column = null;
      String completoNombreCol = (String) iter.next();
      StringTokenizer st = new StringTokenizer(completoNombreCol, ".", false);
      int tokens = st.countTokens();
      if (tokens == 2) { // solo tabla y nombre de columna, sin esquema ni catalogo

         int idx1 = completoNombreCol.indexOf(".");
         tablaId = completoNombreCol.substring(0, idx1).trim();
         tabla = tablaId;
      } else if (tokens == 3) { // no se proporciona el catálogo

         int idx1 = completoNombreCol.indexOf(".");
         int idx2 = completoNombreCol.lastIndexOf(".");
         tablaId = completoNombreCol.substring(0, idx2).trim();
         tabla = completoNombreCol.substring(idx1 + 1, idx2).trim();
         esquema = completoNombreCol.substring(0, idx1).trim();
      } else if (tokens == 4) { // catalogo, esquema, tabla y columna

         tablaId = completoNombreCol.substring(0, completoNombreCol.lastIndexOf(".")).trim();
         int i = 0;
         while (st.hasMoreTokens() && i < 3) {
```

```
            String token = st.nextToken().trim();
            if (i == 0) {
               catalogo = token;
            } else if (i == 1) {
               esquema = token;
            } else if (i == 2) {
               tabla = token;
            }
            i++;
         }
      } else {
         throw new IllegalArgumentException("Mapeo BBDD no valido");
      }
      this.setNombreCatalogo(columnaIdx, catalogo);
      this.setNombreEsquema(columnaIdx, esquema);
      this.setNombreTabla(columnaIdx, tabla);
      this.setNombreColumna(columnaIdx, completoNombreCol.substring(completoNombreCol.lastIndexOf(".") + 1));
      this.setCompletoNombreColumna(columnaIdx, completoNombreCol);
      String nombCampo = (String) mapping.get(completoNombreCol);
      this.setNombreCampo(columnaIdx, nombCampo);
      String[] tablasDistintas = new String[3];
      tablasDistintas[0] = catalogo;
      tablasDistintas[1] = esquema;
      tablasDistintas[2] = tabla;
      this.tablasMap.put(tablaId, tablasDistintas);
      this.setSoloLectura(columnaIdx, false);
      //this.setGrabable(columnaIdx, true); //ver setSoloLectura() method
      this.llenarLeerMethod(columnaIdx, nombCampo);
      this.llenarGrabarMethod(columnaIdx, nombCampo);
      this.setEtiquetaColumna(columnaIdx, toEtiqueta(getNombreColumna(columnaIdx)));
      columnaIdx++;
   }
}

public int getIndiceColumnaPorNombreCampo(String nombreCampoBuscado) {
   if (nombreCampoBuscado == null) {
      return -1;
   }
   for (int i = 0; i < this.nombreCampo.length; i++) {
      if (nombreCampo[i] == null) {
         return -1;
      }
      if (nombreCampoBuscado.equalsIgnoreCase(nombreCampo[i])) {
         return i + 1;
      }
   }
   return -1;
}

public String toEtiqueta(String nombreColumna) {
   String etiqueta = ((String) nombreColumna).toLowerCase();
   StringBuffer sb = new StringBuffer();
   boolean toUpper = false;
   for (int i = 0; i < etiqueta.length(); i++) {
      char c = etiqueta.charAt(i);
      if (c != '_') {
         if (i == 0) {
            toUpper = true;
         }
         if (toUpper) {
            c = Character.toUpperCase(c);
         } else {
         }
         sb.append(c);
         toUpper = false;
      } else {
         if (i > 0) {
            sb.append(' ');
         }
         toUpper = true;
      }
   }
   return sb.toString();
}

@Override
public String toString() {
   StringBuffer sb = new StringBuffer("fpuoc ============ org.fpuoc.Metadatos: ");
   sb.append("\ncontadorColumna: " + contadorColumna);
   sb.append("\nnombreCatalogo: " + Arrays.asList(nombreCatalogo));
   sb.append("\nnombreEsquema: " + Arrays.asList(nombreEsquema));
```

```
        sb.append("\nnombreTabla: " + Arrays.asList(nombreTabla));
        //sb.append("\ncolumnaNombreClass: " + Arrays.asList(columnaNombreClass));
        sb.append("\netiquetaColumna: " + Arrays.asList(etiquetaColumna));
        sb.append("\nnombreColumna: " + Arrays.asList(nombreColumna));
        sb.append("\ncompletoNombreColumna: " + Arrays.asList(completoNombreColumna));
        sb.append("\nnombreCampo: " + Arrays.asList(nombreCampo));
        sb.append("\nleerMethod: " + Arrays.asList(leerMethod));
        sb.append("\ngrabarMethod: " + Arrays.asList(grabarMethod));
        //sb.append("\ngrabable: " + Arrays.asList(grabable));
        sb.append("\ncampoNombreClass: " + Arrays.asList(campoNombreClass));
        sb.append("\nsolicitante: " + solicitante);
        return sb.toString();
    }

    public boolean isThrowExcepcionCampoAusente() {
        return throwExcepcionCampoAusente;
    }

    public void setThrowExcepcionCampoAusente(boolean throwExcepcionCampoAusente) {
        this.throwExcepcionCampoAusente = throwExcepcionCampoAusente;
    }
}
```

Clase: ObjetoPersistente.java

```
package org.fpuoc;

public interface ObjetoPersistente extends java.io.Serializable, Cloneable {

    public static final char ORIGINAL = 'O';
    public static final char CREADO = 'C';
    public static final char ALMACENADO = 'S';
    public static final char ELIMINADO = 'R';
    public static final char INDEFINIDO = 'N';

    public void setValoresOriginales(ContenedorDatos valoresOriginales);
    public ContenedorDatos getValoresOriginales();
    public void setValorOriginal(String nombreCampo, Object valor);
    public Object getValorOriginal(String nombreCampo);
    public char getEstadoPersistencia();
    public void setEstadoPersistencia(char estadoPersistencia);
    public int getObjetoId();
    public void setObjetoId(int objetoId);
    public boolean almacenar();
    public boolean eliminar();
    public boolean crear();
    public boolean cargar();
    public boolean isCargado();
    public void setCargado(boolean cargado);
    public boolean isCargando();
    public void setCargando(boolean cargando);
    public boolean isModificado();
    public void setModificado(boolean modificado);
    public void setMostrarValor(String campo, Object valorMostrado);
    public Object getMostrarValor(String campo);
    public boolean syncOriginal();
    public void setPersistenteFactoryId(long persistenteFactoryId);
    public long getPersistenteFactoryId();
}
```

Clase: ObjetoPersistenteAbstracto.java

```
package org.fpuoc;

import java.beans.*;
import java.util.*;
import java.lang.reflect.*;

/**
 * Generalmente, para conseguir que un objeto sea persistente será necesario
 * que se herede de esta clase.
 * En el caso de que por necesidades de implementación no se pueda heredar
 * de esta clase, deberá implementar la interfaz org.fpuoc.ObjetoPersistente
 * y copiar todos los elementos y métodos desde
 * org.fpuoc.ObjetoPersistenteAbstracto en su propia clase.
 */

public abstract class ObjetoPersistenteAbstracto implements ObjetoPersistente, java.io.Serializable, Cloneable {

   protected boolean modificado = false;
   protected Map valoresMostrados;
   protected static Object[] ARG_LECT_VACIO = new Object[0];
   protected long persistenteFactoryId = -1;
   protected String charset = "UTF-8";
   protected ContenedorDatos valoresOriginales = null;
   protected int objetoId = 0;
   protected char estadoPersistencia = INDEFINIDO;
   protected boolean cargado = false;
   protected boolean cargando = false;


   @Override
   public String toString() {
      StringBuffer sb = new StringBuffer();
      Object valor = null;
      Method[] metodos = getClass().getMethods();
      for (int i = 0;i < metodos.length;i++){
         String nombreMetodo = metodos[i].getName();
         if (nombreMetodo.equals("getValoresComoHTML") || nombreMetodo.equals("getClass")){
            continue;
         }
         if ((nombreMetodo.startsWith("get") || nombreMetodo.startsWith("is")) && metodos[i].getParameterTypes().length == 0){
            try{
               valor = leerValor(metodos[i]);
            }catch (Throwable t){
               continue;
            }
            String fieldCaracterInicial = "";
            if (nombreMetodo.startsWith("is")){
               fieldCaracterInicial = nombreMetodo.substring(2, 3).toLowerCase();
               sb.append(fieldCaracterInicial);
               sb.append(nombreMetodo.substring(3));
            }else if (nombreMetodo.startsWith("get")){
               fieldCaracterInicial = nombreMetodo.substring(3, 4).toLowerCase();
               sb.append(fieldCaracterInicial);
               sb.append(nombreMetodo.substring(4));
            }
            sb.append("=");
            sb.append((valor == null)?"":valor);
            sb.append("&");
         }
      }
      sb.append("valoresMostrados=" + (valoresMostrados == null?"":valoresMostrados.toString()));
      return sb.toString();
   }

   @Override
   public void setValoresOriginales(ContenedorDatos valoresOriginales) {
      this.valoresOriginales = valoresOriginales;
   }

   @Override
   public ContenedorDatos getValoresOriginales() {
      return this.valoresOriginales;
   }

   @Override
   public void setValorOriginal(String nombreCampo, Object valor) {
```

```
    if (valoresOriginales == null) {
      throw new IllegalArgumentException("setValoresOriginales: ContenedorDatos no esta listo");
    }
    valoresOriginales.setObjeto(nombreCampo, valor);
  }

  @Override
  public Object getValorOriginal(String nombreCampo) {
    Object value = (valoresOriginales == null)? null:valoresOriginales.getObjeto(nombreCampo);
    return value;
  }

  @Override
  public char getEstadoPersistencia(){
    return estadoPersistencia;
  }

  @Override
  public void setEstadoPersistencia(char estadoPersistencia){
    this.estadoPersistencia = estadoPersistencia;
  }

  /** Leer la propiedad de objetoId.
   * @return Value of property objetoId.
   *
   */
  @Override
  public int getObjetoId() {
    return objetoId;
  }

  /**
   * Cada vez que FactoriaObjetoPersistente se carga este objeto se le asigna
   * un objetoId que es válido solamente para el mismo
   * FactoriaObjetoPersistente (vea persistenteFactoryId).
   * Si el identificador objetoId de ObjetoPersistente se usa con otro
   * FactoriaObjetoPersistente es preciso conseguir un nuevo identificador
   * objetoId cuando se use con DomainObjectFactory.
   * [setObjeto(...), cargar(), crear(), almacenar(), eliminar()
   */

  @Override
  public void setObjetoId(int objetoId) {
    this.objetoId = objetoId;
  }

  /**
   * Solo sirve para marcar la modificación de datos, la modificación en curso se encuentra en
   * FactoriaObjetoPersistente --> GrabarDatos
   * Normalmente este método es llamado desde FactoriaObjetoPersistente.[crear(ObjetoPersistente), almacenar(ObjetoPersistente),
eliminar(ObjetoPersistente)]
   * @return Boolean para marcar la modificación en curso
   */
  @Override
  public boolean almacenar() {
    if (valoresOriginales != null && valoresOriginales.getContadorCampos() > 0){
      setEstadoPersistencia(ALMACENADO);
      return true;
    }
    return false;
  }

  @Override
  public boolean eliminar() {
    if (valoresOriginales != null && valoresOriginales.getContadorCampos() > 0){
      setEstadoPersistencia(ELIMINADO);
      return true;
    }
    return false;
  }

  @Override
  public boolean crear() {
    if (valoresOriginales == null || valoresOriginales.getContadorCampos() < 1){
      setEstadoPersistencia(CREADO);
      return true;
    }
    return false;
  }
```

```
/**
 * rellenar valoresOriginales con valores desde ObjetoPersistente.
 */
@Override
public boolean syncOriginal(){
   boolean sync = false;
   if (valoresOriginales != null && valoresOriginales.getContadorCampos() > 0){
      try{
         setEstadoPersistencia(ORIGINAL);
         for(int fieldIndex = 1; fieldIndex <= valoresOriginales.getContadorCampos(); fieldIndex++){
            String nombreCampo = valoresOriginales.getNombreCampo(fieldIndex);
            PropertyDescriptor pd = new PropertyDescriptor(nombreCampo, this.getClass());
            Object obj = leerValor(pd.getReadMethod());
            valoresOriginales.setObjeto(fieldIndex, obj);
         }
         sync = true;
         setModificado(false);
      }catch (Throwable t){
         throw new RuntimeException(t);
      }
   }
   return sync;
}

/**
 * rellenar ObjetoPersistente con valores desde valoresOriginales.
 */
@Override
public boolean cargar() {
   setCargando(true);
   if (valoresOriginales != null && valoresOriginales.getContadorCampos() > 0){
      try{
         List listParam = new ArrayList(1);
         for(int fieldIndex = 1; fieldIndex <= valoresOriginales.getContadorCampos(); fieldIndex++){
            listParam.clear();
            String nombreCampo = valoresOriginales.getNombreCampo(fieldIndex);
            Object obj = valoresOriginales.getObjeto(fieldIndex);
            PropertyDescriptor pd = new PropertyDescriptor(nombreCampo, this.getClass());
            listParam.add(obj);
            escribeValor(pd.getWriteMethod(), listParam.toArray());
         }
      }catch(Throwable t){
         throw new RuntimeException(t);
      }
      setCargando(false);
      setCargado(true);
      setModificado(false);
      return true;
   }
   setCargando(false);
   return false;
}

/** Leer la propiedad de cargado.
 * @return Value of property cargado.
 *
 */
@Override
public boolean isCargado() {
   return cargado;
}

/** Asignar nueva propiedad a cargado.
 * @param cargado New valor of property cargado.
 *
 */
@Override
public void setCargado(boolean cargado) {
   this.setEstadoPersistencia(ORIGINAL);
   this.cargado = cargado;
}

/**
 *  Éste es el método más conveniente para usar con JSP.
 *  Debe asegurarse de no llamar a este método desde otro método
 *  de este mismo objeto (o subclase) que empiece con 'get',
 *  o se conseguirá entrar en un bucle interminable.
 */

public String getValoresComoHTML() {
```

```
    StringBuffer sb = new StringBuffer();
    Object obj = null;
    Method[] methods = getClass().getMethods();
    for (int i = 0;i < methods.length;i++){
      String nombreMethod = methods[i].getName();
      if (nombreMethod.equals("getValoresComoHTML") || nombreMethod.equals("getClass")){
        continue;
      }
      if ((nombreMethod.startsWith("get") || nombreMethod.startsWith("is")) && methods[i].getParameterTypes().length == 0){
        try{
          obj = leerValor(methods[i]);
        }catch (Throwable t){
          continue;
        }
        String primerCharCampo = "";
        if (nombreMethod.startsWith("is")){
          primerCharCampo = nombreMethod.substring(2, 3).toLowerCase();
          sb.append(primerCharCampo);
          sb.append(nombreMethod.substring(3));
        }else if (nombreMethod.startsWith("get")){
          primerCharCampo = nombreMethod.substring(3, 4).toLowerCase();
          sb.append(primerCharCampo);
          sb.append(nombreMethod.substring(4));
        }
        sb.append("=");
        if (obj == null){
          sb.append("");
        }else{
          try{
            sb.append(java.net.URLEncoder.encode(obj.toString(), charset));
          }catch (java.io.UnsupportedEncodingException uee){
            throw new RuntimeException(uee);
          }
        }
        sb.append("&");
      }
    }
    sb.deleteCharAt(sb.length() - 1);
    return sb.toString();
  }

  /**
   * Indicador de que este objeto se encuentra en proceso de carga
   * de los valores desde la BBDD
   */

  @Override
  public boolean isCargando() {
    return this.cargando;
  }

  @Override
  public void setCargando(boolean cargando) {
    this.cargando = cargando;
  }

  /**
   * Indicador de que este objeto ha sido modificado.
   * No se llama automaticamente, debe ser invocado por el programador
   *
   */
  @Override
  public boolean isModificado() {
    return this.modificado;
  }

  @Override
  public void setModificado(boolean modificado) {
    this.modificado = modificado;
  }

  /**
   *  Muestra el valor para cada campo en particular. Por ejemplo:
   *  el valor del campo es "123" pero la visualización será "XYZ"
   */
  @Override
  public void setMostrarValor(String campo, Object valorMostrar){
    if (valoresMostrados == null) valoresMostrados = new HashMap();
    valoresMostrados.put(campo, valorMostrar);
```

```
    }

    @Override
    public Object getMostrarValor(String nombreCampo){
        Object mostrar;
        try{
            if (valoresMostrados == null){
                PropertyDescriptor pDesc = new PropertyDescriptor(nombreCampo, this.getClass());
                mostrar = leerValor(pDesc.getReadMethod());
            }else{
                mostrar = valoresMostrados.get(nombreCampo);
            }
        }catch (Throwable t){
            throw new RuntimeException(t);
        }
        return mostrar;
    }

    @Override
    public void setPersistenteFactoryId(long persistenteFactoryId){
        this.persistenteFactoryId = persistenteFactoryId;
    }

    @Override
    public long getPersistenteFactoryId(){
        return this.persistenteFactoryId;
    }

    protected void escribeValor(Method metodo, Object[] valor) throws Throwable{
        try{
            metodo.invoke(this, valor);
        }catch (Throwable t){
            if (t instanceof InvocationTargetException){
                throw ((InvocationTargetException) t).getTargetException();
            }else{
                throw t;
            }
        }
    }

    protected Object leerValor(Method metodo) throws Throwable{
        Object valor = null;
        try{
            valor = metodo.invoke(this, ARG_LECT_VACIO);
        }catch (Throwable t){
            if (t instanceof InvocationTargetException){
                throw ((InvocationTargetException) t).getTargetException();
            }else{
                throw t;
            }
        }
        return valor;
    }
}
```

Clase: Parametros.java

```
package org.fpuoc;

import java.sql.*;

public class Parametros implements ParameterMetaData, java.io.Serializable{

   // parametros - el primer parametro es 1, el segundo es 2, ...
   protected Object[] parametros;
   protected int[] parametroModo;
   protected int[] parametroTipo;
   protected String[] parametroNombreTipo;
   protected int[] precision;
   protected int[] escala;
   protected int[] tipoNull;
   protected boolean[] firmado;
   protected String[] parametroNombre;
   protected int parametroContador;

   public Parametros() {
   }

   @Override
   public boolean isWrapperFor(Class iface){
      return false;
   }

   @Override
   public Class unwrap(Class iface){
      return null;
   }

   public Parametros(int parametroContador) {
      setParams(new Object[parametroContador]);
   }

   protected void init(){
      this.parametroContador = this.parametros.length;
      this.parametroModo = new int[parametroContador];
      this.parametroTipo = new int[parametroContador];
      this.parametroNombreTipo = new String[parametroContador];
      this.precision = new int[parametroContador];
      this.escala = new int[parametroContador];
      this.tipoNull = new int[parametroContador];
      this.firmado = new boolean[parametroContador];
      this.parametroNombre = new String[parametroContador];
   }

   public Parametros(Object[] parametros) {
      setParams(parametros);
   }

   @Override
   public String getParameterClassName(int indice) throws SQLException {
     return parametros[indice - 1].getClass().getName();
   }

   @Override
   public int getParameterCount() throws SQLException {
      return parametroContador;
   }

   @Override
   public int getParameterMode(int indice) throws SQLException {
      return parametroModo[indice - 1];
   }

   public void setParameterMode(int indice, int parametroModo) throws SQLException {
      this.parametroModo[indice - 1] = parametroModo;
   }

   @Override
   public int getParameterType(int indice) throws SQLException {
      return parametroTipo[indice - 1];
   }
```

```
public void setParameterType(int indice, int parametroTipo) throws SQLException {
   this.parametroTipo[indice - 1] = parametroTipo;
}

@Override
public String getParameterTypeName(int indice) throws SQLException {
   return parametroNombreTipo[indice - 1];
}

public void setParameterTypeName(int indice, String parametroNombreTipo) throws SQLException {
   this.parametroNombreTipo[indice - 1] = parametroNombreTipo;
}

@Override
public int getPrecision(int indice) throws SQLException {
   return precision[indice - 1];
}

public void setPrecision(int indice, int precision) throws SQLException {
   this.precision[indice - 1] = precision;
}

@Override
public int getScale(int indice) throws SQLException {
   return escala[indice - 1];
}

public void setScale(int indice, int escala) throws SQLException {
   this.escala[indice - 1] = escala;
}

@Override
public int isNullable(int indice) throws SQLException {
    return tipoNull[indice - 1];
}

public void setNullable(int indice, int tipoNull) throws SQLException {
   this.tipoNull[indice - 1] = tipoNull;
}

@Override
public boolean isSigned(int indice) throws SQLException {
   return firmado[indice - 1];
}

public void setSigned(int indice, boolean firmado) throws SQLException {
   this.firmado[indice - 1] = firmado;
}

public Object[] getParams() {
   return parametros;
}

public void setParams(Object[] parametros) {
   this.parametros = parametros;
   init();
}

public Object getParameter(int indice) {
   return parametros[indice - 1];
}

public Object getParameter(String parametroNombre) throws SQLException {
   return parametros[getParameterIndex(parametroNombre) - 1];
}

public void setParameter(int indice, Object valor) {
  this.parametros[indice - 1] = valor;
}

public String getParameterName(int indice) throws SQLException {
   return parametroNombre[indice - 1];
}

public void setParameterName(int indice, String parametroNombre) throws SQLException {
   if (parametroNombre == null || parametroNombre.trim().length() == 0){
      throw new SQLException("Nombre de parametro no existe");
   }
   parametroNombre = parametroNombre.trim();
   for (int i = 0;i < parametroContador;i++){
```

```
            String nombre = this.parametroNombre[i];
            if (nombre != null && nombre.equalsIgnoreCase(parametroNombre)){
               throw new SQLException("Nombre parametro duplicado: " + parametroNombre);
            }
         }
         this.parametroNombre[indice - 1] = parametroNombre;
      }

      public int getParameterIndex(String parametroNombre) throws SQLException {
         if (parametroNombre == null || parametroNombre.trim().length() == 0){
            return -1;
         }
         for (int i = 0;i < parametroContador;i++){
            String nombre = this.parametroNombre[i];
            if (nombre != null && nombre.equalsIgnoreCase(parametroNombre)){
               return i + 1;
            }
         }
         return -1;
      }

      @Override
      public String toString(){
         StringBuffer sb = new StringBuffer();
         for (int i = 0;i < parametroContador;i++){
            sb.append("\nparam").append(i + 1).append(": ");
            sb.append("&parametroNombre=").append(parametroNombre[i]);
            sb.append("&valor=").append(parametros[i]);
            sb.append("&parametroModo=").append(parametroModo[i]);
            sb.append("&parametroTipo=").append(parametroTipo[i]);
            sb.append("&parametroNombreTipo=").append(parametroNombreTipo[i]);
            sb.append("&precision=").append(precision[i]);
            sb.append("&escala=").append(escala[i]);
            sb.append("&tipoNull=").append(tipoNull[i]);
            sb.append("&firmado=").append(firmado[i]);
         }
         return sb.toString();
      }
   }
```

Clase: PKCache.java

```
package org.fpuoc;

import java.util.*;
import java.sql.Connection;

/**
 * Recupera y almacena PrimaryKeys desde java.sql.DatabaseMetaData o entrada del usuario
 */

public class PKCache implements java.io.Serializable, Cloneable {

   private PKCache() {
      if (Boolean.getBoolean("debug-fpuoc") || Boolean.getBoolean("debug-" + getClass().getName())){
         setDebug(true);
      }
   }

   private static PKCache instancia = null;
   /* Clave por cada tabla */
   protected Map pkCache = new HashMap();
   protected boolean debug = false;
   private static final String DOT = ".";


   public static PKCache getInstance() {
      if (instancia == null) {
         instancia = new PKCache();
      }
      return instancia;
```

```
  }

  /*
   * Este método permite poner columnas para una tabla de PrimaryKey
   * sin conexión a la base de datos, etcétera.
   *
   * Asegurese de usar el formato:  catalogo + DOT + esquema + DOT + tabla + DOT + columna
   * o bien el formato: esquema + DOT + tabla + DOT + columna
   */
  public synchronized void setPrimaryKey(String catalogo, String esquema, String tabla, List pk){
     String tablaId = null;
     if (catalogo != null && catalogo.trim().length() != 0){
        tablaId = catalogo + DOT + esquema + DOT + tabla;
     }else if (esquema != null && esquema.trim().length() != 0){
        tablaId = esquema + DOT + tabla;
     }else if (tabla != null && tabla.trim().length() != 0){
        tablaId = tabla;
     }else{
        throw new IllegalArgumentException("setPrimaryKey(): desaparecidos todos los identificadores de tabla");
     }
     if (pk == null || pk.size() == 0){
        throw new IllegalArgumentException("setPrimaryKey(): la lista de PK esta vacia");
     }
     if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::setPrimaryKey()::tablaId: " + tablaId + "
pk: " + pk);
     pkCache.put(tablaId, pk);
  }

  public synchronized List getPrimaryKey(String catalogo, String esquema, String tabla){
     if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::catalogo: " + catalogo
+ "::esquema: " + esquema + "::tabla: " + tabla);
     String tablaId = null;
     //Build tabla id
     if (catalogo != null && catalogo.trim().length() != 0){
        tablaId = catalogo.trim() + DOT + esquema.trim() + DOT + tabla.trim();
     }else if (esquema != null && esquema.trim().length() != 0){
        tablaId = esquema.trim() + DOT + tabla.trim();
     }else if (tabla != null && tabla.trim().length() != 0){
        tablaId = tabla.trim();
     }else{
        throw new IllegalArgumentException("getPrimaryKey(): Todos los identificadores de tabla han desaparecido");
     }
     if(this.pkCache.containsKey(tablaId)){
        return (List) this.pkCache.get(tablaId); //ya está almacenado (caché)
     }else{
        return null;
     }
  }

  public synchronized List getPrimaryKey(Connection conexion, String catalogo, String esquema, String tabla){
     if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::catalogo: " + catalogo
+ "::esquema: " + esquema + "::tabla: " + tabla);
     String tablaId = null;
     //Build tabla id
     if (catalogo != null && catalogo.trim().length() != 0){
        tablaId = catalogo.trim() + DOT + esquema.trim() + DOT + tabla.trim();
     }else if (esquema != null && esquema.trim().length() != 0){
        tablaId = esquema.trim() + DOT + tabla.trim();
     }else if (tabla != null && tabla.trim().length() != 0){
        tablaId = tabla.trim();
     }else{
        throw new IllegalArgumentException("getPrimaryKey(): Todos los identificadores de tabla han desaparecido");
     }
     List pkCampos = (List) this.pkCache.get(tablaId);
     if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::tablaId: " + tablaId + "
pkFields: " + pkCampos);
     if (pkCampos != null) {
        return pkCampos; //ya está almacenado (caché)
     }else{
        pkCampos = new ArrayList();
     }
     java.sql.ResultSet pkInfo = null;
     try{
        java.sql.DatabaseMetaData dbmd = conexion.getMetaData();
        if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this
+"::getPrimaryKey()::DatabaseMetaData: " + dbmd.getDatabaseProductName() + "::catalogo: " + catalogo + "::esquema: " + esquema + "::tabla: " +
tabla);
        pkInfo = dbmd.getPrimaryKeys(catalogo, esquema, tabla);
        if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::pkInfo: " + pkInfo);
        while(pkInfo.next()){
           if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::pkInfo.next()");
```

```
            String nombreColumna = pkInfo.getString(4);
            pkCampos.add(tablaId + DOT + nombreColumna);
          }
          if (debug) System.out.println("fpuoc ============ " + new Date() + " " + this.getClass() + "::" + this +"::getPrimaryKey()::pkCampos: " +
pkCampos);
          if (pkCampos.isEmpty()){
            throw new java.sql.SQLException("No se puede recuperar informacion de la PrimaryKey desde DatabaseMetaData");
          }
        }catch (java.sql.SQLException sqle){
          throw new RuntimeException(sqle);
        }catch (Exception e){
          throw new RuntimeException(e);
        }finally{
          try{
            if (pkInfo != null) pkInfo.close();
          }catch (java.sql.SQLException sqle){
            sqle.printStackTrace();
            throw new RuntimeException(sqle);
          }
        }
        pkCache.put(tablaId, pkCampos);
        return pkCampos;
    }

    public boolean isDebug() {
        return debug;
    }

    public void setDebug(boolean debug) {
        this.debug = debug;
    }
}
```

Clase: Transformador.java

```
package org.fpuoc;

public interface Transformador {
    public java.sql.ResultSet getResultSet();
    public void setResultSet(java.sql.ResultSet resultSet);
    public Object[] transformar(String campoNombreClase, String nombreColumna) throws java.sql.SQLException;
}
```

Clase: ValorObjeto.java

```
package org.fpuoc;

public class ValorObjeto implements Cloneable, java.io.Serializable, Comparable{

    protected Object valor;
    protected String valorEtiqueta;
    protected boolean compararPorValor = false;
    protected boolean errorCompararConNull;

    public ValorObjeto() {}

    public ValorObjeto(Object valor, String valorEtiqueta) {
        this.valor = valor;
        this.valorEtiqueta = valorEtiqueta;
    }

    public ValorObjeto(Object valor, String valorEtiqueta, boolean compararPorValor) {
        this.valor = valor;
        this.valorEtiqueta = valorEtiqueta;
        this.compararPorValor = compararPorValor;
    }

    public Object getValor() {
```

```
    return valor;
  }

  public void setValor(Object valor) {
    this.valor = valor;
  }

  public String getValorEtiqueta() {
    if (valorEtiqueta == null){
      if (valor != null){
        valorEtiqueta = valor.toString();
      }else{
        valorEtiqueta = "";
      }
    }
    return valorEtiqueta;
  }

  public void setValorEtiqueta(String valorEtiqueta) {
    this.valorEtiqueta = valorEtiqueta;
  }

  public boolean isCompararPorValor() {
    return compararPorValor;
  }

  public void setCompararPorValor(boolean compararPorValor) {
    this.compararPorValor = compararPorValor;
  }

  @Override
  public String toString(){
    return getValorEtiqueta();
  }

  @Override
  public int compareTo(Object objeto) {
    if (objeto == null) {
      if (errorCompararConNull){
        throw new NullPointerException("ValorObjeto es null");
      }else{
        return -1;
      }
    }
    if (this.equals(objeto)){
      return 0;
    }
    ValorObjeto vo = (ValorObjeto) objeto;
    int i1 = this.valorEtiqueta.compareTo(vo.valorEtiqueta);
    int i2 = ((Comparable) this.valor).compareTo((Comparable) vo.valor);
    if (i1 > 0 && i2 > 0){
      return 1;
    }else if (i1 < 0 && i2 < 0){
       return -1;
    }else{
      if (compararPorValor){
        return i2;
      }else{
        return i1;
      }
    }
  }

  @Override
  public boolean equals(Object objeto) {
    if (!(objeto instanceof ValorObjeto)) {
      return false;
    }
    ValorObjeto vo = (ValorObjeto) objeto;
    return ((valor == null?vo.valor == null:valor.equals(vo.valor)) && (valorEtiqueta == null?vo.valorEtiqueta ==
null:valorEtiqueta.equals(vo.valorEtiqueta)));
  }

  @Override
  public int hashCode() {
    return ((valor == null?-1:valor.hashCode()) ^ (valorEtiqueta == null?-1:valorEtiqueta.hashCode()));
  }

  public boolean isErrorCompararConNull() {
    return errorCompararConNull;
  }
```

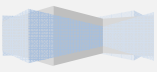
```
    public void setErrorCompararConNull(boolean errorCompararConNull) {
        this.errorCompararConNull = errorCompararConNull;
    }

}
```

Clase: TransformadorDefecto.java

```
package org.fpuoc;

import java.io.Serializable;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TransformadorDefecto implements Serializable, Transformador {

    protected transient ResultSet resultSet;

    public TransformadorDefecto() {
    }

    public TransformadorDefecto(ResultSet resultSet) {
        setResultSet(resultSet);
    }

    @Override
    public Object[] transformar(String campoNombreClase, String nombreColumna) throws SQLException{
        Object valor = null;
        Object[] convertirValor = null;
        /**
         * Solo se han incorporado los DataTypes.
         * Se considera que en una ampliacion posterior se puedan
         * incorporar más tipos nuevos
         */
        if (campoNombreClase == null){
            convertirValor = new Object[1];
            valor = resultSet.getObject(nombreColumna);
            convertirValor[0] = valor;
        }else if (campoNombreClase.equals("String")){
            convertirValor = new String[1];
            valor = resultSet.getString(nombreColumna);
            if (resultSet.wasNull()){
                convertirValor[0] = (String) null;
            }else{
                convertirValor[0] = valor;
            }
        }else if (campoNombreClase.equals("Integer")){
            convertirValor = new Integer[1];
            valor = new Integer(resultSet.getInt(nombreColumna));
            if (resultSet.wasNull()){
                convertirValor[0] = (Integer) null;
            }else{
                convertirValor[0] = valor;
            }
        }else if (campoNombreClase.equals("Long")){
            convertirValor = new Long[1];
            valor = new Long(resultSet.getLong(nombreColumna));
            if (resultSet.wasNull()){
                convertirValor[0] = (Long) null;
            }else{
                convertirValor[0] = valor;
            }
        }else if (campoNombreClase.equals("Double")){
            convertirValor = new Double[1];
            valor = new Double(resultSet.getDouble(nombreColumna));
            if (resultSet.wasNull()){
                convertirValor[0] = (Double) null;
            }else{
                convertirValor[0] = valor;
            }
        }else if (campoNombreClase.equals("Float")){
            convertirValor = new Float[1];
```

```
    valor = new Float(resultSet.getFloat(nombreColumna));
    if (resultSet.wasNull()){
      convertirValor[0] = (Float) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("Short")){
    convertirValor = new Short[1];
    valor = new Short(resultSet.getShort(nombreColumna));
    if (resultSet.wasNull()){
      convertirValor[0] = (Short) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.sql.Timestamp")){
    convertirValor = new java.sql.Timestamp[1];
    valor = (java.sql.Timestamp) resultSet.getTimestamp(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.sql.Timestamp) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.sql.Time")){
    convertirValor = new java.sql.Time[1];
    valor = (java.sql.Time) resultSet.getTime(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.sql.Time) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.sql.Date")){
    convertirValor = new java.sql.Date[1];
    valor = (java.sql.Date) resultSet.getDate(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.sql.Date) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.util.Date")){
    convertirValor = new java.util.Date[1];
    java.sql.Date fecha = resultSet.getDate(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.util.Date) null;
    }else{
      valor = new java.util.Date(fecha.getTime());
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.math.BigDecimal")){
    convertirValor = new java.math.BigDecimal[1];
    valor = resultSet.getBigDecimal(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.math.BigDecimal) null;
    }else{
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("java.math.BigInteger")){
    convertirValor = new java.math.BigInteger[1];
    long valorLong = 0;
    valorLong = resultSet.getLong(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = (java.math.BigInteger) null;
    }else{
      valor = java.math.BigInteger.valueOf(valorLong);
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("boolean")){
    convertirValor = new Boolean[1];
    boolean valorBoolean = false;
    valorBoolean = resultSet.getBoolean(nombreColumna);
    if (resultSet.wasNull()){
      convertirValor[0] = new Boolean(false);
    }else{
      valor = new Boolean(valorBoolean);
      convertirValor[0] = valor;
    }
  }else if (campoNombreClase.equals("byte")){
    convertirValor = new Byte[1];
    byte valorByte = 0;
    valorByte = resultSet.getByte(nombreColumna);
    if (resultSet.wasNull()){
      byte byteVacio = 0;
```

```
            convertirValor[0] = new Byte(byteVacio);
        }else{
            valor = new Byte(valorByte);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("char")){
        convertirValor = new Character[1];
        String valorString = null;
        valorString = resultSet.getString(nombreColumna);
        if (resultSet.wasNull()){
            char charVacio = '\u0000';
            convertirValor[0] = new Character(charVacio);
        }else{
            valor = new Character(valorString.charAt(0));
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("double")){
        convertirValor = new Double[1];
        double valorDouble = 0.0;
        valorDouble = resultSet.getDouble(nombreColumna);
        if (resultSet.wasNull()){
            convertirValor[0] = new Double(0.0);
        }else{
            valor = new Double(valorDouble);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("float")){
        convertirValor = new Float[1];
        float valorFloat = 0;
        valorFloat = resultSet.getFloat(nombreColumna);
        if (resultSet.wasNull()){
            convertirValor[0] = new Float(0.0);
        }else{
            valor = new Float(valorFloat);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("int")){
        convertirValor = new Integer[1];
        int valorInt = 0;
        valorInt = resultSet.getInt(nombreColumna);
        if (resultSet.wasNull()){
            convertirValor[0] = new Integer(0);
        }else{
            valor = new Integer(valorInt);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("long")){
        convertirValor = new Long[1];
        long valorLong = 0;
        valorLong = resultSet.getLong(nombreColumna);
        if (resultSet.wasNull()){
            convertirValor[0] = new Long(0);
        }else{
            valor = new Long(valorLong);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("short")){
        convertirValor = new Short[1];
        short valorShort = 0;
        valorShort = resultSet.getShort(nombreColumna);
        if (resultSet.wasNull()){
            short s = 0;
            convertirValor[0] = new Short(s);
        }else{
            valor = new Short(valorShort);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("Boolean")){
        convertirValor = new Boolean[1];
        boolean valorBoolean = false;
        valorBoolean = resultSet.getBoolean(nombreColumna);
        if (resultSet.wasNull()){
            convertirValor[0] = new Boolean(false);
        }else{
            valor = new Boolean(valorBoolean);
            convertirValor[0] = valor;
        }
    }else if (campoNombreClase.equals("Character")){
        convertirValor = new Character[1];
```

```
            String valorString = null;
            valorString = resultSet.getString(nombreColumna);
            if (resultSet.wasNull()){
               char emptyChar = '\u0000';
               convertirValor[0] = new Character(emptyChar);
            }else{
               valor = new Character(valorString.charAt(0));
               convertirValor[0] = valor;
            }
         }else{
            convertirValor = new Object[1];
            valor = resultSet.getObject(nombreColumna);
            convertirValor[0] = valor;
         }
         return convertirValor;
      }

      @Override
      public ResultSet getResultSet() {
         return resultSet;
      }

      @Override
      public void setResultSet(ResultSet resultSet) {
         this.resultSet = resultSet;
      }
   }
```