**Framework de Persistencia en Java:**

**Hibernate.**

***Alumno: Mariela Lannia***

***Profesor: Leonel Guccione***

***Materia: Seminario***

***Carrera: TSSI***

***Universidad: UTN MDP***

**Índice**

Introducción…………………………………………………….2

Alcances y Limitaciones………………………………………..3

Marco Teórico……………………………………………….….4

ORM……………………………………………………4

Persistencia de Objetos…………………………………5

Entonces, ¿Qué es Hibernate?.........................................5

Arquitectura de Hibernate……………………………...6

El fichero de Configuración de Mapeo………………...6

Sesiones Hibernate……………………………………..8

Funciones Principales…………………………………9

El Lenguaje de Consulta……………………………...10

Ciclo de vida del Objeto Persistente………………….10

El Entorno de Ejecución……………………………...11

La Aplicación………………………………………………...12

Diagrama de Clases…………………………………………..15

Diagrama de Componentes…………………………………..16

Ventajas y Desventajas de Hibernate………………………...17

Conclusiones………………………………………………….18

Recomendaciones…………………………………………….19

Agradecimientos……………………………………………...20

Bibliografia……………………………………………….......21

**Introducción**

Trabajar con software orientado a objetos en Java y con una base de datos relacional puede llegar a ser muy repetitivo en cuanto al código de conexión a los datos almacenados y puede llegar a tomar bastante tiempo en los entornos empresariales de hoy en día.

¿Nos preguntamos si no hay alguna herramienta que nos permita facilitar este trabajo?

Así bien llegamos a nuestra hipótesis que se baza en que el modelo ORM (Mapeo Objeto – Relacional) Hibernate, permite desarrollar aplicaciones de manera sencilla, de fácil mantenimiento y con un nivel de independencia respecto del modelo de datos.

Por lo tanto la presente investigación tiene como objetivo el estudio de la persistencia de objetos Java en una base de datos relacional, utilizando para ello una de las soluciones disponibles para esta integración denominada ORM (Mapeo Objeto – Relacional) con una de sus herramientas mas conocidas llamada Hibernate, en donde la persistencia se realiza de forma totalmente automática y transparente a la lógica de la aplicación. También se incluye el análisis, diseño e implementación de un sistema ejemplo de Gestión de Información Facultativa. Todo este desarrollo se enmarca en un ámbito de ejecución Web con requerimientos clásicos de una aplicación administrativa.

Hibernate es un marco de trabajo Java que proporciona mecanismos de mapeo objeto/relacional para definir cómo se almacenan, eliminan, actualizan y recuperan los objetos Java. No solamente se ocupa del mapeo desde las clases Java a las tablas de las bases de datos (y desde los tipos de datos de Java a los tipos de datos de SQL), sino que también facilita la consulta y recuperación de datos. Esto puede reducir de manera importante el tiempo de desarrollo que se tomaría con el manejo de datos de forma manual en SQL y JDBC.

**Alcances y limitaciones**

Los logros esperados son partiendo de los conocimientos básicos obtenidos en el manejo de obtención de datos con una base de datos relacional, lograr un manejo de la herramienta de mapeo con sus funciones, características y fundamentos básicos. Para contar con una base solida sobre una nueva herramienta muy común a la hora de facilitar las tareas de los programadores.

Para concluir con nuestra investigación, nuestra meta es desarrollar un sistema utilizable y ejemplar, que cuente con capacidad de almacenar, recuperar información de una base de datos y lograr la persistencia de los objetos mapeados, utilizando la herramienta de ORM Hibernate para el mapping entre los dos modelos (orientado a objetos y relacional), con el fin de probar los beneficios que ofrecen los ORM principalmente reducir la complejidad de la aplicación y el tiempo de desarrollo

Además, a lo largo de la investigación y desarrollo del sistema hemos avanzado sobre herramientas de implementación tales como TableSorter y JasperReports que facilitaron la creación de nuevas funciones importantes agregadas al mismo.

El software a diseñar como un ejemplo ilustrador de nuestra investigación será de forma local, se descarta la opción de futuras instalaciones en un servidor web ya que el mismo solo se realizara en forma de ejemplo y método de aplicación de aprendizaje.

**Marco Teórico**

**ORM**

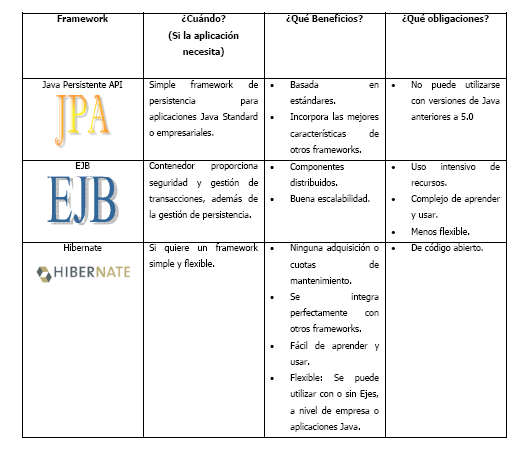
El mapeo objeto/relacional (ORM) es la automatización y manejo transparente de la persistencia de objetos de una aplicación en Java a las tablas de una base de datos relacional.

Un ORM en esencia se encarga de transformar los datos de una representación a otra, es decir, realiza el marco de trabajo de la persistencia debido a que sabe como consultar la base de datos para recuperar objetos Java, independientemente de saber como persistir los objetos en la representación de la base de datos. El mapeo de los meta datos típicamente es en archivos XML.

Existes varias alternativas de herramientas de ORM disponibles, entre ellas source y comerciales.

Entre las más populares de open source encontramos principalmente Hibernate del Apache DB Project y el las comerciales principalmente Top Link de Oracle.

Tipos de ORM:

## Persistencia de Objetos

La tarea de persistir objetos Java en una base de datos relacional actualmente está siendo facilitada por un gran número de herramientas que permiten a los desarrolladores dirigir motores de persistencia para convertir objetos Java a columnas/registros de una base de datos y viceversa. Esta tarea implica serializar objetos Java estructurados en forma de árbol a una base de ratos relacional estructurada de forma tabular y viceversa. Esencial para este esfuerzo es la necesidad de mapear los objetos Java a columnas y registros de la base de datos de una manera optimizada en velocidad y eficiencia.

El marco de trabajo Hibernate se enfrenta al problema "objeto-java-a-base-de-datos" de forma tan elegante como cualquier otro marco de trabajo disponible. Funciona persistiendo y restaurando viejos objetos Java (POJOs) utilizando un modelo de programación muy transparente y poco exigente.

## Entonces, ¿Que es Hibernate?

Hibernate es un marco de trabajo Java que proporciona mecanismos de mapeo objeto/relacional para definir cómo se almacenan, eliminan, actualizan y recuperan los objetos Java. Además, ofrece servicios de consulta y recuperación que pueden optimizar los esfuerzos de desarrollo dentro de entornos SQL y JDBC. Por último, Hibernate reduce el esfuerzo necesario para convertir hojas de resultados de la base de datos relacional en gráficos de objetos Java.

Una de las características únicas de Hibernate es que no requiere que los desarrolladores implementen interfaces propietarios o extiendan clases bases propietarias para poder persistir las clases. En vez de eso, Hibernate trata con la *reflexión* de Java y el aumento de clases en tiempo de ejecución utilizando una librería de generación de código Java muy poderosa y de alto rendimiento llamada **CGLIB**. CGLIB se utiliza para extender clases Java e implementar interfaces Java en tiempo de ejecución.

Para la mayoría de las aplicaciones, almacenar y recuperar información implica alguna forma de interacción con una base de datos relacional. Esto ha representado un problema fundamental para los desarrolladores porque algunas veces el diseño de datos relacionales y los ejemplares orientados a objetos comparten estructuras de relaciones muy diferentes dentro de sus respectivos entornos.

Las bases de datos relacionales están estructuradas en una configuración tabular y los ejemplares orientados a objetos normalmente están relacionados en forma de árbol. Esta 'diferencia de impedancia' ha llevado a muchos de nosotros, desarrolladores, a intentar construir un puente entre el mundo relacional y el mundo orientado a objetos.

**Arquitectura de Hibernate**

En una visión general, Hibernate presenta dos requerimientos indispensables para el mapeo entre los dos modelos.

El primer requerimiento son las propiedades. Se refiere con propiedades al archivo de configuración que contiene información sobre que base de datos se desea usar, el driver a utilizar para la comunicación, el puerto, el usuario, contraseña, factoría de transacciones, los recursos de mapeo, etc. Es decir, toda la información necesaria para la conexión con la base de datos

Se declara como muy flexible y conveniente almacenar la configuración en un fichero llamado hibernate.cfg.xml. Este fichero se sitúa en la raíz del classpath del contexto de la aplicación (por ejemplo: WEB-INF/classes). Se puede acceder a este fichero utilizando la clase net.sf.hibernate.cfg.Configuration en tiempo de ejecución.

El siguiente código muestra una configuración típica de este fichero:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/seminario</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">1234</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.current\_session\_context\_class">thread</property>

</session-factory>

</hibernate-configuration>

**El Fichero de Configuración de Mapeo**

Para que Hibernate se encargue del mapeo de las clases en Java es recomendable hacer un diseño de JavaBeans.

Por convención los JavaBeans contienen dos métodos para cada uno de sus correspondientes atributos, específicamente son los métodos públicos getter y setter. Todos los atributos de la clase son privados, aunque Hibernate puede acceder a los métodos de la clase private, public o protected pero esto le daría un poco mas de robustez al diseño para el refactoring. El constructor vació es necesario para las clases persistentes debido a que Hibernate será el encargado de la creación de los objetos.

Las aplicaciones Hibernate hacen uso de ficheros de mapeo que contienen metadatos que definen los mapeos objeto/relacional para las clases Java. Un fichero de mapeo tiene el sufijo .hbm.xml. Dentro de cada fichero de configuración, se mapean a tablas de la base de datos las clases que se van a persistir, las cuales definimos su estructura básica en el párrafo anterior, y las propiedades se definen con mapeos de campo/columna y claves primarias.

Ejemplo:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!-- Generated 04-dic-2012 22:15:57 by Hibernate Tools 3.2.1.GA -->

<hibernate-mapping>

<class catalog="seminario" name="Mapeo.Categorias" table="categorias">

<id name="idcategorias" type="int">

<column name="idcategorias"/>

<generator class="assigned"/>

</id>

<property name="nombre" type="string">

<column length="45" name="nombre"/>

</property>

<property name="esAlum" type="java.lang.Integer">

<column name="EsAlum"/>

</property>

<set inverse="true" name="empleadoses">

<key>

<column name="emp\_categoria\_id"/>

</key>

<one-to-many class="Mapeo.Empleados"/>

</set>

<set inverse="true" name="alumnoses">

<key>

<column name="al\_categoria\_id"/>

</key>

<one-to-many class="Mapeo.Alumnos"/>

</set>

</class>

</hibernate-mapping>

El elemento <class> especifica la clase java(bean) de la que se esta hacienda el mapping y la table de la base de datos a relacionar en el mapeo.

El elemento <id> relaciona la clave primaria. Opcionalmente se puede especificar el tipo (type) de dato con el que se trabaja. En caso de no estar especificado Hibernate lo podría deducir.

El sub-elemento <generator> permite definir cómo se generarán los valores de las llaves primarias. Hibernate ofrece diversos métodos predefinidos para la generación.

El elemento <property> define las propiedades que no forman parte de la clave primaria, el atributo unique permite definir si se permitirán valores duplicados o no, asi como el atributo not-null permite especificar si la propiedad puede tener el valor null.

El elemento <many-to-one > describe una relación de muchos a uno entre la misma clase. Se debe especificar el campo de la tabla que contiene la clave primaria para ser relacionado.

**Sesiones Hibernate**

Las interfaces de Session, Transaction y Query son llamadas por la aplicación para realizar básicamente operaciones de ABM y ejecución de queries. Especialmente Session es la principal interface entre java e Hibernate ya que mantiene la comunicación entre la base de datos y la aplicación. Transacción permite a la aplicación definir unidades de trabajo y es asociada con una Session en particular. Por otro lado Query es una instancia obtenida por la Session, que nos permite ejecutar consultas con facilidad definiendo el número máximo de resultados a ser devueltos, especificar el primer registro que se desea obtener, etc. Estas interfaces son la principal dependencia para el control de la aplicación.

Pada poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto Session utilizando la clase SessionFactory de Hibernate. El siguiente fragmento de código ilustra este proceso:

// Crea la session factory

SessionFactory factory = cfg.buildSessionFactory();

// Obtain the new session object

Session session = factory.openSession();

Una vez inicializada la configuración, se puede hacer cualquier modificación adicional de forma programática. Sin embargo, estas modificaciones se deben hacer antes de crear el ejemplar de SessionFactory.

Normalmente, el ejemplar de SessionFactory sólo se crea una vez y luego se utiliza para crear todas las sesiones relacionadas con un contexto dado.

Un objeto Session Hibernate representa una única unidad-de-trabajo para un almacén de datos dado y lo abre un ejemplar de SessionFactory. Se deben cerrar las sesiones cuando se haya completado todo el trabajo de una transacción.

Después de tener acceso a las propiedades y la configuración, se puede empezar a insertar, borrar, actualizar y consultar.

**Funciones Principales**

* **INSERTAR**

Para hacer persistente un objeto hay varias maneras de hacerlo, nosotros usaremos la siguiente:

Se crea una instancia del objeto a guardar (1), se crea la sesión (2), se inicia la unidad de trabajo (3), se guarda el objeto (4), se confirma si la transacción se realizo de manera exitosa (5) (sino deberemos hacer un rollback), cerrar la sesión (6).

Session sesion =HibernateUtil.getSessionFactory().getCurrentSession();

sesion.beginTransaction();

sesion.save(nuevoAlumno);

sesion.getTransaction().commit();

* **BORRAR**

El esquema general para borrar un objeto de la base de datos es básicamente el mismo que el de insertar cambiando el método llamado de sesión por:

sesion.delete(nuevoAlumno);

En el caso de que se desee borrar más de un objeto esta tarea se puede realizar con una consulta en el lenguaje Hibernate Query Languege (HQL) que ya veremos mas adelante.

* **ACTUALIZAR**

En caso de querer hacer actualizaciones de los datos, se podrá hacer obteniendo un objeto desde la session, seteando alguno de sus parámetros y utilizando el método update() que nos brinda session.

Por ejemplo:

Alumnos alumno = (Alumnos) session.load(Alumno.class,Integer(id);

alumno.setAlNombre(‘Victoria’);

session.update(alumno);

**El Lenguaje de Consultas de Hibernate**

Hibernate ofrece un lenguaje de consultas que agrupa un potente y flexible mecanismo de consulta, almacenamiento, actualización y recuperación de objetos desde una base de datos. Este lenguaje, el *Hibernate Query Language* (**HQL**), es una extensión orientada a objetos de SQL. HQL permite acceder a los datos de varias formas, incluyendo consultas orientadas a objetos, como en el método find() del siguiente ejemplo:

List alumnos = session.find("from Alumnos as e order by e.departamentos.depNombre",Hibernate.STRING );

Si se prefiere, se puede utilizar SQL o expresar una consulta SQL, utilizando createSQLQuery():

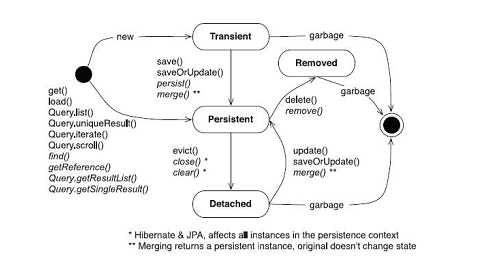
Query q = sesion.createQuery(txt);

lista = (List<Alumnos>) q.list();

El HQL proporcionado por Hibernate es capaz de soportar clausulas como por ejemplo: group by,count,and,avg,sum,max,min, entre otras.

**Ciclo de Vida de Objetos Persistentes**

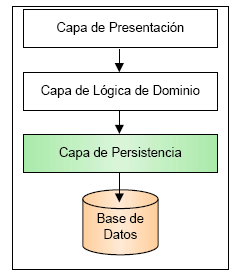
Ya que Hibernate es un mecanismo que permite dar solución a la capa de persistencia de un modo transparente, los objetos que son persistidos no se deben preocupar de los estados por los que pasa. Es Hibernate el que maneja estos estados. Se definen 4 estados, ocultando de la complejidad de su implementación al usuario del motor.



**El Entorno de Ejecución**

**Arquitectura basada en capas**

Para organizar una aplicación empresarial, la industria del software ha convergido en una Arquitectura basada en Capas, dividiendo el sistema en tres capas básicas: la capa de presentación, la capa de dominio y la capa de persistencia, como se muestra en la siguiente figura.



Cada capa solo depende de ella y de la que se encuentre abajo.

**Capa de Presentación**: Maneja la interacción entre el usuario y la aplicación.

**Capa de Lógica de Dominio:** Representa conceptos de negocio como reglas o estados.

**Capa de Persistencia:** Esta capa brinda servicios para sincronizar la capa de lógica con un medio de almacenamiento.

**La Aplicación**

Una vez detallados los conceptos básicos, solo nos resta enfocarnos en el análisis de los requerimientos del software a desarrollar. Incluyendo una descripción general del sistema y sus

Funciones principales:

* Almacenar y recuperar datos de una base de datos relacional con código Java, utilizando la herramienta de ORM Hibernate para el mapeo entre los dos módulos.
* Con la técnica anterior constituir un control de usuarios para limitar el uso del sistema.
* Armar listados con los datos almacenados que permitan un orden aleatorio según lo requiera el cliente.
* Guardar los listados de información en archivos PDF para su posterior impresión o manipulación brindándole al cliente la posibilidad de elegir donde serán guardados y con que nombre.

**Descripción de interfaces entre los Frameworks**

Pasaremos a describir la interacción entre los componentes de nuestro sistema.

El contexto web es proporcionado a través de GlassFish Server 3 siguiendo una arquitectura cliente-servidor.

* Interacción con el ORM:

La interacción será constante con el ORM Hibernate debido a que a este le será encargando el manejo de la persistencia de datos para facilitar el desarrollo de la aplicación.

* Interacción con la base de datos:

La interfaz del programa con la base de datos será el ORM Hibernate versión 3.0, por lo tanto la interacción será constante al igual que el recién mencionado, y el Sistema Gestionador de Base de Datos (SGBD) será MySQL.

* Interacción con el servidor de Aplicaciones Web:

El sistema a implementar esta enfocado a un ambiente web.

Para implementar especificaciones de servlets y JSPs requerimos un servidor de aplicaciones y un contenedor de servlets. GlassFish Server es la herramienta que utilizaremos para cumplir con estos requerimientos.

Diseñaremos una sencilla aplicación Web que permite crear, borrar y recuperar datos de alumnos, empleados y departamentos de un ambiente facultativo, utilizando un navegador Web. Las peticiones de cliente se pasarán desde un navegador a un servlet Java, que comunica con un servicio de usuario, que comunica con accesos a los objetos de datos basados en Hibernate .

**Componentes utilizados sobre el desarrollo**

Deberemos contar con los siguientes componentes para el desarrollo de nuestro sistema:

* NetBeans 6.9.1 : Plataforma de desarrollo.
* Hibernate 3.0
* GlassFish Server 3: Glassfish es un programa contenedor web, que sigue el estandar JavaEE2. Sirve principalmente para traducir código java (servlets) a código HTML y mantener una comunicación entre el servidor y el cliente.

Si bien en un principio se pensó en utilizar Apache Tomcat 5, se cambio a este motor simplemente por disponibilidad directa del dispositivo.

Se agregar librería proporcionadas por Netbeans al proyecto.

* MySQL Server 5.2: Este es el componente que utilizaremos como servidor de base de datos. Sera el encargado de almacenar los datos persistentes de la aplicación. Es fundamental contar con un driver conector de Java a MySQL.

Agregaremos mysql-connector-java-5.0.7-bin.jar a los archivos jar del proyecto.

* Jasper Reports: Es una herramienta gratuita que esta compuesta de un conjunto de librerías Java para facilitar la generación de informes en nuestra aplicación.

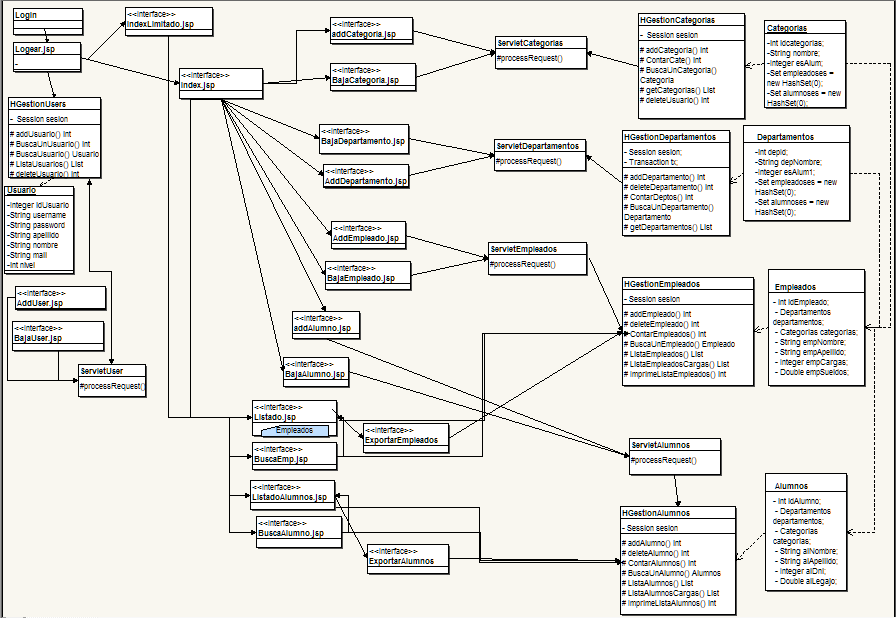
Se agregaran las librerías detalladas a continuación al proyecto de Netbeans:

* jasperreports-3.7.6.jar
* jasperreports-3.7.6-javaflow.jar
* jasperreports-fonts-3.7.6.jar
* commons-beanutils-1.8.0.jar
* commons-collections-2.1.1.jar
* commons-digester-1.7.jar
* commons-logging-1.0.4.jar
* commons-javaflow-20060411.jar
* groovy-all-1.7.5.jar
* iText-2.1.7.jar
* png-encoder-1.5.jar
* poi-3.6.jar
* IReports: Programa por el cual crearemos nuestros reportes para luego llenarlo desde la aplicación. Los reportes generados deberán posteriormente copiarse en el disco raíz C:\\ .
* Java EE (Java Enterprise Edition): Plataforma Java basada en componentes intercambiables que residen en un servidor de aplicaciones y son gestionados por él. Ofrece un framework para el desarrollo de aplicaciones distribuidas en multicapas basada en Web.

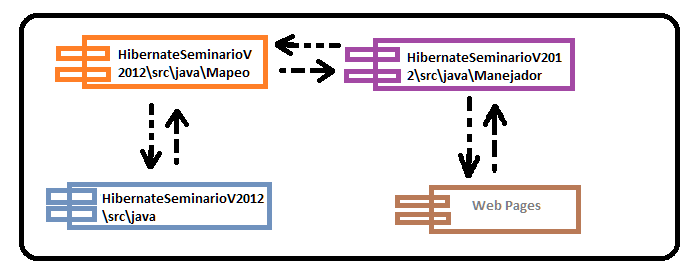
Se agrega archivo .jar llamado javaee.jar al proyecto en Netbeans.

* Servlets: Los servlets, son objetos que corren dentro del contexto de un servidor de aplicaciones o también conocido como contenedor de servlets y extienden su funcionalidad.  
  El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Diagrama de Clases



**Diagrama de Componentes**



* **HibernateSeminarioV\src\java\Mapeo**

Este paquete contiene todos archivos que definen las características del mapeo. Incluyendo fundamentalmente un archivo fundamental llamado HibernateUtil.java quien será el encargado de crear las sesiones.

* **HibernateSeminarioV\src\java**

En este paquete por defecto solo se alojaran dos archivos de gran importancia. Uno de ellos será el hibernate.cfg.xml que contiene la configuración de Hibernate detallada anteriormente.

El otro será un archivo de ayuda para la creación de los objetos mapeados llamado hibernate.reveng.xml quien básicamente es el encargado de tomando la configuración del configurador crear las clases mapeados con sus correspondientes archivos xml.

* **HibernateSeminarioV\src\java\Majedor**

Contiene todos los servlets y los archivos de gestión de los objetos persistentes. Desde aquí es donde se llamaran todos los procesos que se volcaran en la persistencia de los datos final.

* **Web Pages**

Este paquete contiene todos los archivos jsp que estarán encargados de la capa de presentación del sistema, es decir, el diseño visual del mismo.

**Ventajas y Desventajas del uso de Hibernate**

**Ventajas**

**Total independencia con el motor** **de base de datos** ya que tenemos totalmente independiente la capa de datos con la capa de lógica de negocio.  Por tanto bastará con cambiar una línea de un fichero de conexión y ya tendremos la aplicación rodando bajo otro motor de base de datos.

La ventaja más importante podríamos decir que es que se **utilizan datos cacheados**.  Los datos cacheados son objetos residentes en memoria principal que son rellenados al acceder por primera vez desde Hibernate, esto quiere decir que los sucesivos accesos serán de coste ínfimo ya que ya están los datos cargados en memoria.

Facilidad de programación ya que la orientación a objetos facilita muchísimo el pasar de un diagrama a código fuente.

**Mantenibilidad**: Al tener pocas líneas de código permite que el código sea más comprensible.

**Rendimiento**: Existe la tendencia a pensar que una solución “manual” es más eficiente que una “automática”. Hay que tener en cuenta que una solución automática, permite que dediquemos más tiempo a optimizaciones.

El uso de aplicaciones de persistencia de objetos en bases de datos relacionales es cada vez común. En el mercado existen muchos Framework dedicados a esta tarea que hacen el trabajo de almacenamiento de objetos de una forma transparente para el programador.

**Desventajas**

No apto para aplicaciones de gran manejo de datos ya que el mapeo podría ser insostenible en memoria.

No es el método mas clásico de enseñanza, ni se cuenta con mucha información sobre el uso del mismo, por lo tanto a nivel personal una desventaja es que hay que dedicar mucho tiempo en su aprendizaje.

## Conclusiones

Luego de un intenso estudio y aplicación del mismo, basándonos en conocimientos y practicas con bases de datos relacionales. Llegamos a la conclusión de que nuestra hipótesis fue **Correcta.**

La técnica de persistencia una vez que se ha puesto en uso, no solo determinara una mejora en el rendimiento de aplicaciones web sino que hará menor la tarea del programador a la hora de obtener, modificar o borrar datos de la base de datos relacional y el mantenimiento de estos será simplemente cambiando solo líneas de código especificas.

Según nuestra investigación la técnica ORM será la mejor opción para desarrollar un software, ya que podremos contar con las características beneficiarias del manejo de objetos y aprovechar la madurez y estandarización de las base de datos relacionales.

Las diferencias arquitecturales entre el árbol de objetos Java y las tablas de las bases de datos relacionales, hacen que sea un poco desalentadora para los desarrolladores la tarea de persistir objetos Java en una base de datos relacional. Hibernate define un mecanismo de mapeo objeto/relacional y un lenguaje de consulta que hace que el almacenamiento y recuperación desde un almacén de datos sea una proposición verdaderamente sencilla cuando se conoce en profundidad.

**Recomendaciones**

Es fundamental antes de comenzar conocer en profundidad la herramienta que usaremos, ya que de esto dependerán las raíces del diseño y la base de nuestra aplicación además del rendimiento de la misma.

También se recomienda tener un diseño y visión futura de las tablas necesarias para la aplicación ya que una vez realizado el mapeo con la herramienta de reveng.xml proporcionada por Hibernate, si se debe reutilizar esto puede duplicar código que dañara nuestra aplicación.

Hibernate al igual que Servlets es ***case sensitive*** (Sensible a mayúsculas y minúsculas) por lo tanto poner mucha atención principalmente a la hora de hacer alguna Consulta **HQL.**

**Agradecimientos**

En primer lugar a la persona que ahora me cuida desde lejos. Angela Barbieri, es para vos este esfuerzo Nonna.

Lic. Victoria Sánchez Peña, principalmente por la gran motivación e incentivación, y el apoyo en todo este año de dedicación. Convirtiéndose en gran pilar de este proyecto.

María Ana F. Calise y al resto de mi familia, por el aguante en los momentos más difíciles.

Héctor Garro, con el apoyo y acompañamiento continúo.

Leonel Guccione, por su tiempo y enseñanza.

Gonzalo Benoffi, por sus consejos y orientación en un momento decisivo.

A todas las personas de la UTN que me ayudaron a rendir este pendiente después de varios años y varias idas y vueltas.

A todos mis Amigos y Amigas que con sus palabras de apoyo me acompañaron todos los días. Principalmente a Romina Battistessa, que estuvo todos estos años a mi lado en las buenas y en las malas.

**Bibliografía**

**Internet**

[http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html\_single/#tutorial-firstapp](http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html_single/)

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

[**http://www.javatutoriales.com/2009/02/creacion-de-reportes-con-jasperrepots-y.html**](http://www.javatutoriales.com/2009/02/creacion-de-reportes-con-jasperrepots-y.html)

[**http://sourceforge.net/projects/jasperreports/files/jasperreports/**](http://sourceforge.net/projects/jasperreports/files/jasperreports/)

[**http://community.jaspersoft.com/search?query=website%20ireportwebsite%20IR%20Website%20ir%20download**](http://community.jaspersoft.com/search?query=website%20ireportwebsite%20IR%20Website%20ir%20download)

[**https://docs.google.com/viewer?a=v&q=cache:toH9TwaQzj4J:www.javahispano.org/storage/contenidos/Seminario\_Hibernate\_3.pdf+&hl=es&gl=ar&pid=bl&srcid=ADGEESj2q8jT\_vpl565QJwTXvkqgCScxE1S508RH78mHSJn19\_ch\_Ydmh0uGRRjTxWDTQEKBIP3XjS8N4HfjgNX-Z\_z\_2vDQ9w2Q9-mwsxQofin6OHdUXBMoGf4Ekh8HRrUkb4s-7Ncr&sig=AHIEtbQ9apeJI360-JzSe8Gp4YxeNhBopw**](https://docs.google.com/viewer?a=v&q=cache:toH9TwaQzj4J:www.javahispano.org/storage/contenidos/Seminario_Hibernate_3.pdf+&hl=es&gl=ar&pid=bl&srcid=ADGEESj2q8jT_vpl565QJwTXvkqgCScxE1S508RH78mHSJn19_ch_Ydmh0uGRRjTxWDTQEKBIP3XjS8N4HfjgNX-Z_z_2vDQ9w2Q9-mwsxQofin6OHdUXBMoGf4Ekh8HRrUkb4s-7Ncr&sig=AHIEtbQ9apeJI360-JzSe8Gp4YxeNhBopw)