Seminario

FRAMEWORK HIBERNATE



Cristeche Ezequiel  
Vazquez Ezequiel

Profesor: Leonel Guccione

Universidad Tecnológica Nacional

Tecnicatura Superior en Sistemas Informáticos

**Índice**

Introducción………………………………………………………………………..1

Alcances y Limitaciones…………………………………………………………....2

Marco de referencia……………………………………………….……………......2

Marco Teórico…………………………………………………………......…….….3

Antes de los ORM…………………………………………………………………..4

Ventajas de los ORM……………………………………………………….………5

Desventaja de los ORM…………………………………………………………….5

Hibernate………………………………………………............................................6

Características…………………………………………………………………….....6

Conceptos básicos de Hibernate………………………………………………….....7

HQL…………………………………………..……………………………………..11

Desarrollo de la aplicación………………………..…………………………………12

Software a entregar……...………………………..…………………………………12

Diagrama de clases.……...………………………..…………………………………13

Conclusiones……...……...………………………..………………………………...14

Bibliografia……...……...……………………..…..………………………………...14

**Introducción**

Cualquier organización con fines de lucro, ya sea dedicada al desarrollo de Software o no, busca optimizar los procesos de producción, maximizar las ganancias y minimizar los costos. Los sistemas desarrollados en las factorías de software deben ser comercialmente atractivos, útil para los usuarios y de calidad.

Para certificar la calidad del software existen las normas ISO, las cuales ofrecen una guía para el desarrollo, implementación y mantenimiento de software. Al estar certificados por ISO se logra una mejor documentación del sistema, se amplía la satisfacción del cliente, se agiliza el tiempo de desarrollo de un sistema y se da una mayor percepción de calidad al mercado. Los clientes ven con buenos ojos los sistemas certificados por las normas ISO, les da confianza.

Ahora bien, ISO nos ofrece una guía de cómo debemos trabajar para ofrecer al mercado un sistema de calidad, partiendo de esta base, podemos utilizar distintas herramientas que agilicen los procesos de documentación, desarrollo, implementación y soporte de un sistema.

Enfocado en el área de desarrollo me planteo:

**¿Que software libre puedo utilizar para optimizar los tiempos en un entorno de desarrollo donde se utiliza Java y bases de datos relacionales para la persistencia de datos?**

Sabemos que la interacción de un programador con la base de datos es constante, ya sea para crear, eliminar o recuperar datos es necesario escribir código SQL. Muchas veces las tareas son repetitivas, por ejemplo los ABM, por cada tipo de objeto que quiero guardar en la base de datos debo crear una clase que permita insertarlo, eliminarlo, modificarlo y consultarlo, con excepción de consultas especiales, el resto es siempre lo mismo. Al utilizar un mapeador Objeto Relacional (ORM), estas tareas se ejecutan automáticamente liberando al programador de escribir código SQL para realizar estas tareas, debiendo sólo escribir las consultas especiales. Hibernate es un ORM gratis para la plataforma java, el cual facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación (en memoria).

**Alcances y limitaciones**

El resultado esperado de este seminario es obtener conocimientos sobre los beneficios y aplicación del ORM Hibernate, utilizado hoy en día en muchas empresas de sistemas. Consideramos oportuno e importante investigar sobre la aplicación de este framework y sobre todo aplicarlo en un sistema que desarrollaremos, donde realmente podremos observar resultados. Este proyecto será más orientado a la aplicación de la herramienta. En esta oportunidad presentaremos un sistema de escritorio, donde se pueda aplicar el ORM mencionado, precisamente será un sistema de fichajes y control de empleados, que puede servir, por ejemplo, para un entorno educativo, o laboral.

**Marco de referencia:**

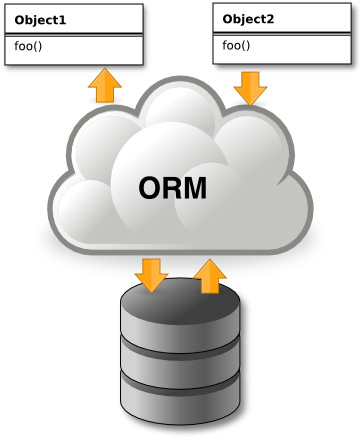
Hibernate es un Framework para persistir objetos en una base de datos relacional, funciona como un mapeador de atributos entre ambos modelos. La idea de este ORM la tuvo Gavin King, Ingeniero actual Grupo de JBoss, el cual cansado de la ineficiencia y complejidad de los sistemas de persistencia de la época, ideo un sistema base que fue apoyado por un inmenso grupo de programadores alrededor del mundo. Este FrameWork es uno de los más utilizados en la actualidad y es de fácil adaptación, libre, posee soporte para java y .net, mucha documentación y un grupo de programadores mundiales que soportan su crecimiento. Hibernate está diseñado para poder adaptarse a su uso sobre una base de datos existente o también posee la funcionalidad de crear la base de datos a partir de la información detallada por el programador y de sus relaciones.

Para el desarrollo de la aplicación de escritorio utilizamos Java como lenguaje de programación, Netbeans como IDE, para la persistencia de datos elegí MySQL, el ORM a implementar será el mencionado Hibernate e implementaré otro Framework llamado OpenSwing, el cuál incluye un conjunto de componentes gráficos y utilizaremos al menos un componente gráfico para demostrar su implementación.

**Marco Teórico:**

**ORM (Mapeo Objeto - Relacional)**

En los sistemas orientados a objetos la información se representa como clases y objetos, mientras que en las bases de datos relacionales, se presenta en forma de tablas relacionales. Por lo tanto, para almacenar la información tratada en un programa orientado a objetos en una base de datos relacional es necesaria una traducción entre ambas formas. El Mapeo Objeto-Relacional se encarga precisamente de eso, de transformar los datos de una representación a otra.



Entre los **sistemas gestores de bases de datos** más utilizados se encuentran:

* **SQL Server de Microsoft:** Existen al menos una docena de ediciones diferentes dirigidas a un público específico, desde la versión Express, gratuita y con limitaciones (1 GB de consumo de memoria y 4 GB de capacidad) para pequeñas aplicaciones hasta las ediciones enterprise para soluciones empresariales.
* **mySQL:** Anteriormente propiedad de Sun Microsystems y actualmente de Oracle, es la base de datos más empleada por la comunidad OpenSource . Posee licencia pública general de GNU.
* **PosgreSQL:** Empleada en ámbito docente aunque no por ello menos potente que la anterior, es libre y mantenida por la comunidad de desarrolladores.
* **Oracle:** Considerado por muchos uno de los sistemas de bases de datos más complejos, usado fundamentalmente en el mercado de servidores empresariales donde su hegemonía ha sido casi total hasta hace poco, por la competencia del resto de sistemas antes mencionados. Posee también una versión Express (gratuita)

# Entre los Frameworks ORM más utilizados se encuentran:

* Hibernate
* NHibertnate
* Linq
* Oracle TopLink
* iPersist
* Next:
* OpenStep:

# Antes de los ORM

Antes de la aparición de estos sistemas las consultas se tenían que realizar a mano dentro de las propias aplicaciones, con lo cual había que crear una petición a la base de datos de manera manual (y específica para cada sistema, ya que no todos los gestores de bases de datos tienen la misma implementación del lenguaje SQL), costando más tiempo en el desarrollo y propenso a errores humanos.

# Ventajas de los ORM

* Rapidez en el desarrollo. La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado.
* Abstracción de la base de datos. Al utilizar un sistema ORM, lo que conseguidos es separarnos totalmente del sistema de Base de datos que utilicemos, y así si en un futuro debemos de cambiar de motor de bases de datos, tendremos la seguridad de que este cambio no nos afectará a nuestro sistema, siendo el cambio más sencillo.
* Reutilización. Nos permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas.
* Seguridad. Los ORM suelen implementar sistemas para evitar tipos de ataques como pueden ser los SQL injections.
* Mantenimiento del código. Nos facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.
* Lenguaje propio para realizar las consultas. Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar la sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta.

# Desventajas de los ORM

* Tiempo utilizado en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
* Aplicaciones algo más lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

 **Hibernate** es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java, que emplea atributos declarativos mediante XML o anotaciones, se distribuye con licencia GNU/LGPL y posee una versión para .NET llamada nHibernate. Permite elegir la Base de datos relacional con la que queremos interactuar (Sql Server, PostGre, MySql, Oracle…).

Como mencionaba anteriormente el mapeo objeto-relacional nos ayudará será precisamente a eso, a olvidarnos completamente de cómo convertir los objetos en datos primitivos  para almacenarlos y viceversa. Genera automáticamente el código Sql usando mapeo Objeto – relacional, el cual se especifica en un documento XML.



**Características de Hibernate:**

* + Muy buena documentación (foros para ayuda, libros)
  + Comunidad activa con muchos usuarios
  + Transacciones, polimorfismo, herencia.
  + Potente lenguaje de consulta (HQL): subconsultas, outer joins.

**Porqué elegir Hibernate?**

* + Hibernate es actualmente el más potente ORM del mercado
  + Es software libre
  + Facilita mucho trabajo en desarrollo de aplicaciones OO con BD relacionales

Hace totalmente

* + transparente el uso de la BD
  + Ahorro en tiempo de desarrollo
  + Evita fallas humanas al escribir consultas SQL directa sobre la base de datos
  + Podemos cambiar de BD sin cambiar líneas de código

**Conceptos básicos de Hibernate**

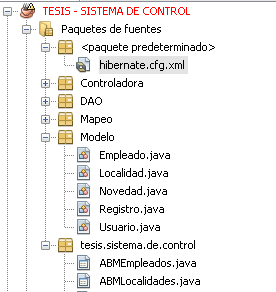
Para almacenar y recuperar estos objetos de la base de datos, el desarrollador debe mantener una conversación con el motor de Hibernate mediante un objeto especial, quizás el concepto clave más importante dentro Hibernate, que es la **Sesión** (**clase Session**). Se puede equiparar a grandes rasgos al concepto de conexión de JDBC y cumple un papel muy parecido, es decir, sirve para delimitar una o varias operaciones relacionadas dentro de un proceso de negocio, demarcar una transacción y aporta algunos servicios adicionales como una caché de objetos para evitar interacciones innecesarias contra la BD. En este sentido veremos que la clase Session ofrece métodos como save(Object object), createQuery(String queryString), beginTransaction(), close(), etc. para interactuar con la BD tal como estamos acostumbrados a hacerlo con una conexión JDBC (de hecho "envuelve" una conexión JDBC), pero con una diferencia: mayor simplicidad, es decir, guardar un objeto, por ejemplo, consiste en algo así como session.save(miObjecto), sin necesidad de especificar una sentencia SQL, y esto es sólo un ejemplo en el que ganamos utilizando Hibernate.

**Nuestros objetos se asocian a la sesión para poder realizar las operaciones de persistencia (Save, Update, Delete).**

Configuración de Hibernate:

La configuración de Hibernate puede hacerse en tres lugares:

* Un archivo de propiedades llamado "hibernate.properties".
* Un archivo XML llamado "hibernate.cfg.xml".
* En código dentro de la misma aplicación.

Utilizaré el archivo "hibernate.cfg", el IDE se encargará de colocarle la extensión ".xml".   
 

En este archivo se configuran los parámetros básicos y típicos para una conexión (la URL, nombre de usuario, contraseña, driver, etc.). Cada uno de estos parámetros se configura dentro de una etiqueta "<property>" (al igual que casi todos los elementos del archivo de configuración). Como dijimos antes, usamos una base de datos MySQL para este ejemplo, así que la configuración queda de la siguiente forma:

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

**<property name="connection.url">jdbc:mysql://localhost/control</property>**

**<property name="connection.username">usuario</property>**

**<property name="connection.password">password</property>**

Después, configuramos el pool de conexiones de Hibernate. En este caso como es un ejemplo muy simple, solo nos interesa tener una conexión en el pool, por lo que colocamos la propiedad "**connection.pool\_size**" con un valor de "1":

**<property name="connection.pool\_size">1</property>**

Otras dos propiedades importantes que podemos configurar son: **"show\_sql"**, que indica si queremos que las consultas SQL generadas sean mostradas en el stdout (normalmente la consola), y **"hbm2ddl.auto",** que indica si queremos que se genere automáticamente el esquema de la base de datos (las tablas). **"show\_sql"** puede tomar valores de **"true"** o **"false"**, nosotros lo colocaremos en **"true"** (lo que puede ser bueno mientras estamos en etapas de desarrollo o pruebas, pero querrán cambiar su valor cuando su aplicación pase a producción). Por otro lado **"hbm2ddl.auto"** puede tomar los valores, según [la documentación oficial](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html), **de "validate", "update", "create", y "create-drop"** (falta "none") (aunque no todos los valores funcionan para todas las bases de datos).

Con el valor "create-drop" hacemos que cada vez que se ejecute la aplicación Hibernate elimine las tablas de la base de datos y las vuelva a crear, cuando su sistema pase a producción sería bueno que quitaran esta propiedad.

Entonces el archivo de configuración queda de la siguiente forma:



En el archivo de mapeo **(Empleado, Registro, Localidad, Usuario, Novedad)** .hbm.xml definimos el mapeo entre los atributos y los campos de la tabla, como así también el identificador y las relaciones con otras tablas.

**EJEMPLO ARCHIVO XML (EMPLEADO. hbm.xml)**



Ahora veremos cómo crear el código de acceso a la base de datos, crearemos una clase de utilidad llamada "**HibernateUtil**", que se hará cargo de inicializar y hacer el acceso al "**SessionFactory**".

La clase "**HibernateUtil"** queda de la siguiente forma:

**import org.hibernate.HibernateException;**

**import org.hibernate.SessionFactory;**

**import org.hibernate.cfg.AnnotationConfiguration;**

**public class HibernateUtil**

**{**

**private static final SessionFactory sessionFactory;**

**static**

**{**

**try**

**{**

**sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();**

**} catch (HibernateException he)**

**{**

**System.err.println("Ocurrió un error en la inicialización de la SessionFactory: " + he);**

**throw new ExceptionInInitializerError(he);**

**}**

**}**

**public static SessionFactory getSessionFactory()**

**{**

**return sessionFactory;**

**}**

**}**

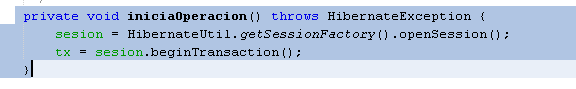
La SessionFactory proporciona instancias del objeto sesión, típicamente hay una  
única SessionFactory para toda la aplicación, creada durante la inicialización de la misma. Sin embargo, si la aplicación accede a varias bases de datos se necesitará una SessionFactory por cada base de datos. La sesión es el canal entre la base de datos y la aplicación, nuestra aplicación necesitará crear y destruir sesiones todo el tiempo, quizá en cada petición. Las instancias de sesión deben ser usadas sólo por una única transacción o unidad de trabajo.

Pada poder utilizar los mecanismos de persistencia de Hibernate se debe obtener un objeto Session utilizando la clase SessionFactory de Hibernate.

**Funciones Principales (INSERT, DELETE, UPDATE)**

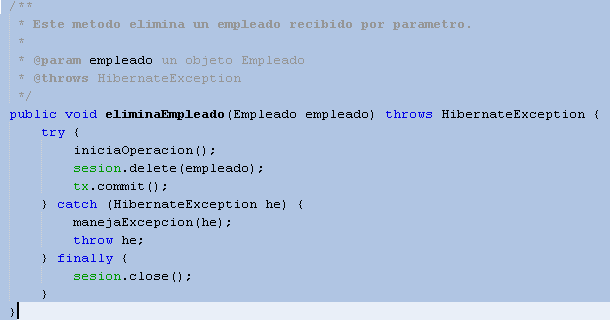
**INSERT:**

Creamos una instancia del objeto a guardar, obtenemos la sesión y e iniciamos una transacción, guardamos el objeto, confirmamos si la transacción se realizó correctamente, por último cerramos la sesión.





**DELETE:**  
Al igual que en el INSERT, en nuestro caso a través del método ***IniciaOperacion*** (Creamos una instancia del objeto a guardar, obtenemos la sesión y e iniciamos una transacción) y le pasamos como parámetro el objeto a ser eliminado. En el caso de que se produzca algun tipo de error a la hora de iniciar la transacción, será atajado por el método ***ManejaExcepcion.***



**HQL – Hibernate Query Language**

El Hibernate Query Languaje (HQL) es el lenguaje de consultas que usa Hibernate para obtener los objetos desde la base de datos. Su principal particularidad es que las consultas se realizan sobre los objetos java que forman nuestro modelo de negocio, es decir, las entidades que se persisten en Hibernate. Ésto hace que HQL tenga las siguientes características:

* *Los tipos de datos son los de Java.*
* *Las consultas son independientes del lenguaje de SQL específico de la base de datos*
* *Las consultas son independientes del modelo de tablas de la base de datos.*
* *Es posible tratar con las colecciones de Java.*
* *Es posible navegar entre los distintos objetos en la propia consulta*.

Las consultas HQL se lanzan (o se ejecutan) sobre el modelo de entidades que hemos definido en Hibernate, esto es, sobre nuestras clases de negocio.

La consulta:

***SELECT \* FROM Localidad where e.id = 1***

podría traducirse como :

***SELECT IDLOCALIDADES, LOCALIDAD from localidades where id =1***

*LOCALIDAD de la primer consulta hacer referencia a la clase y No a la tabla.*

**Software a entregar**Realizaremos un sistema de Gestión y Administración de fichajes de empleados, aplicando las tecnologías anteriormente mencionadas. El mismo, está compuesto por 2 módulos. Sistemas de administración de fichajes y sistema de gestión de fichajes. Nuestra aplicación puede ser utilizado en diferentes ámbitos profesionales. Puede aplicarse a facultades, empresas, ferreterías, salón de eventos, etc. El único requisito a cumplir por parte del cliente, es contar con una cámara que permita agendar los códigos QR generados para los diferentes empleados.  
Nuestro sistema esta compuesto de la siguiente manera:

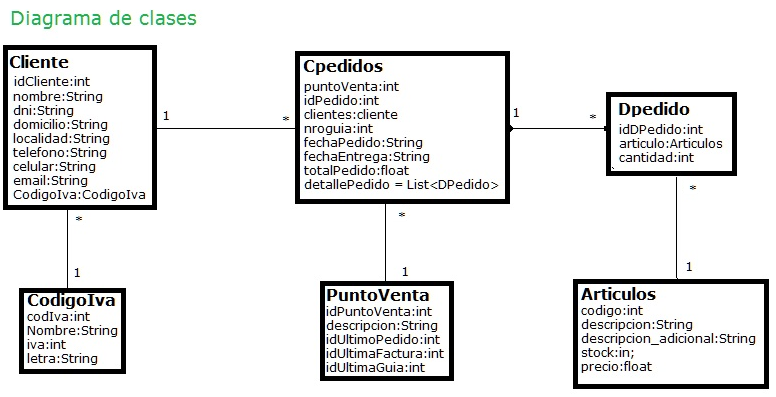
En el sistema se utilizará:

- Alta, baja y modificación de Localidades y de Empleados.

- Alta, baja y modificación de pedidos de clientes.

- Alta, baja y modificación de Guías.

- Visualización del piqueo de las distintas Guías.

****

# Conclusiones

**Ventajas de utilizar hibernate:**

Total independencia con el motor de la base de datos, solo con cambiar una línea del archivo hibernate.cfg ya tendremos la aplicación funcionando con otro motor de base de datos.

Desarrollar utilizando un ORM es más rápido y fácil de mantener porque se reducen la cantidad de líneas.

Al no tener que escribir consultas SQL directas contra el motor de la base de datos se ahorran fallas humanas. Hibernate se encarga de casi toda la persistencia de datos, solo nos tenemos que concentrar en escribir aquellas consutlas específicas.

**Desventajas**

Como principal desventaja me encontré con la configuración y correcta implementación de la herramienta, si bien la documentación sobre hibernate es abundante también está desordenada. En un momento hibernate dejó de funcionar y pensé que el problema estaba en el código y perdí mucho tiempo, reinstalando hibernate comenzó todo a funcionar nuevamente. Hay que dedicarle un tiempo al aprendizaje y a las “mañas” que puede tener utilizar un ORM.

No se puede utilizar para grandes aplicaciones ya que el mapeo podría ser insostenible en memoria.

Entonces… confirmando la hipótesis inicial, Hibernate como herramienta gratuita, correctamente configurado e implementado, mejora el tiempo de producción y trae muchos beneficios, aunque puede llevar un tiempo aprender a utilizarlo correctamente.

**Bibliografía**

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

<http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html_single/>

<http://www.javatutoriales.com>

<http://oswing.sourceforge.net/setup.html>

<https://netbeans.org/kb/docs/java/hibernate-java-se.html>

<http://hibernate.org/>

<http://en.wikipedia.org/wiki/Hibernate_%28Java%29>

http://www.javahispano.org/storage/contenidos/hibernate\_1.pdf‎