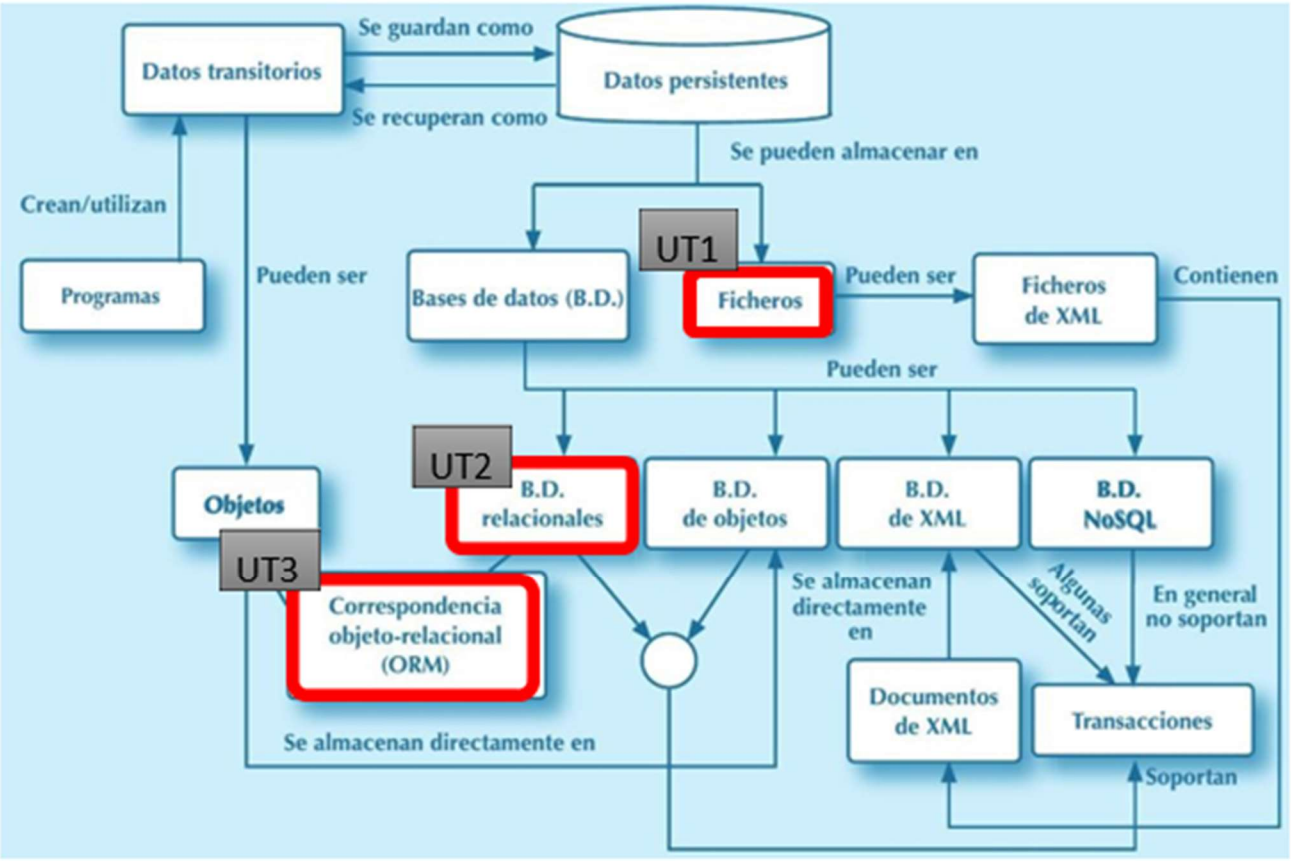
UT3. MAPEO OBJETO-RELACIONAL

Tabla de contenido

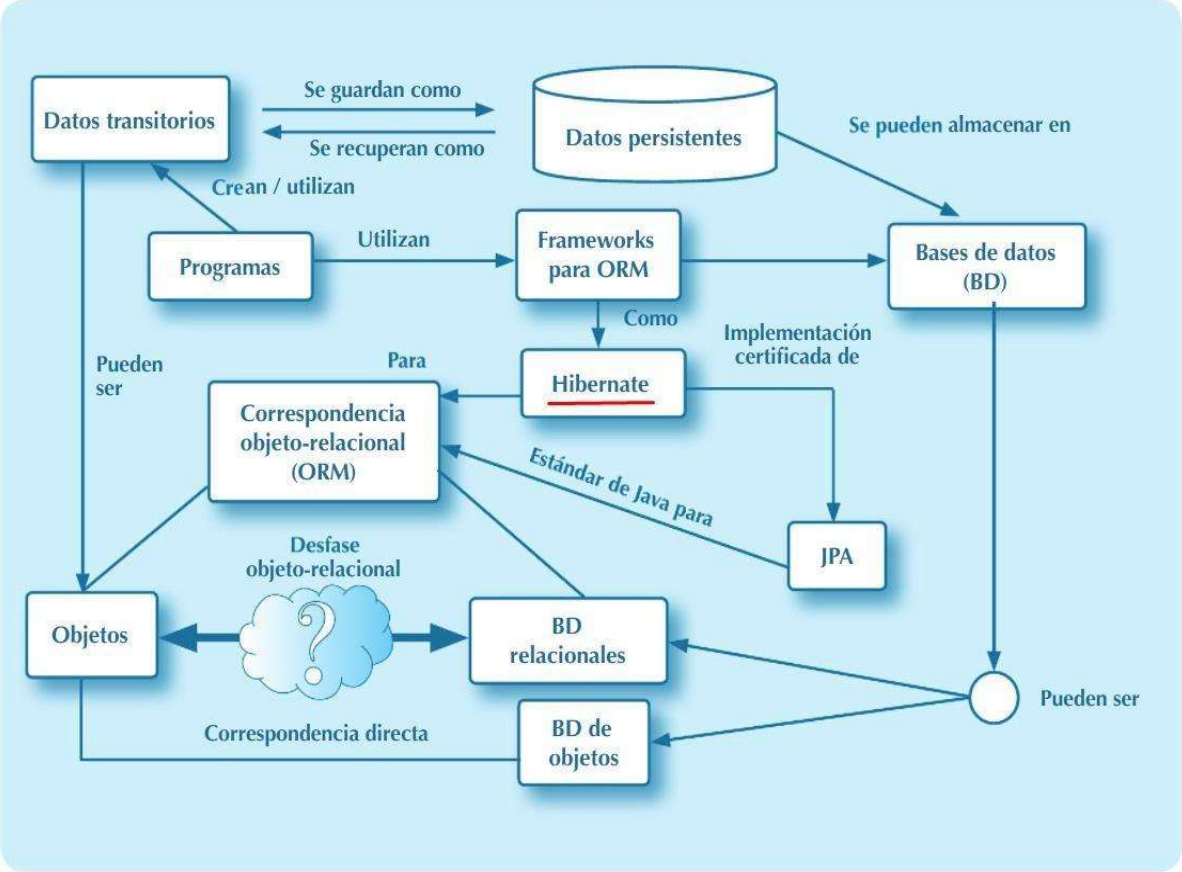
Contenido

1. Introducción

Recordatorio esquema general del módulo

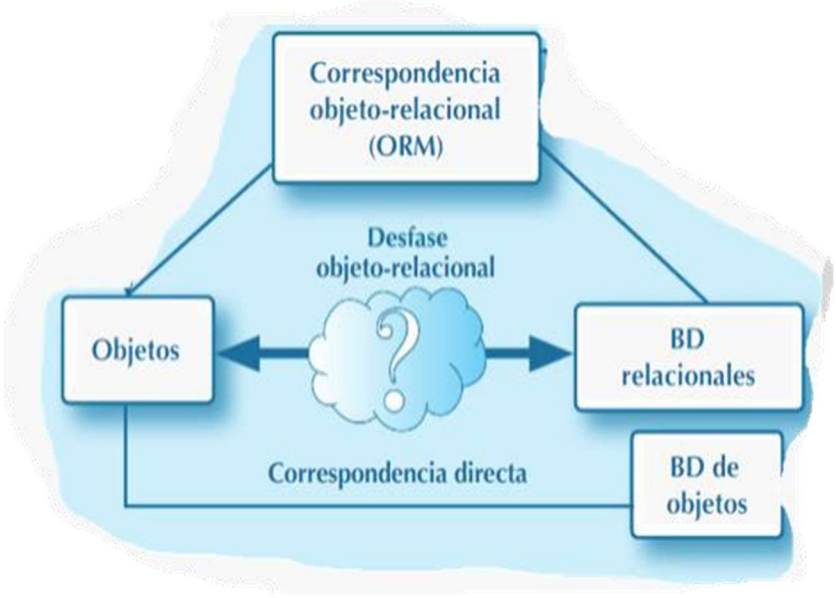


En esta unidad:



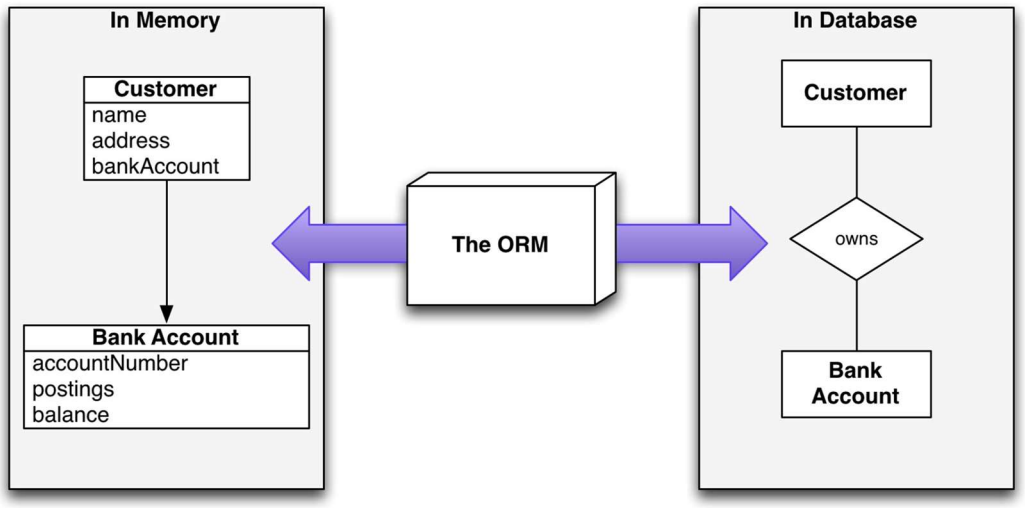
El desfase objeto-relacional

El desfase objeto-relacional (también conocido como impedancia objeto-relacional o Object-Relational Impedance Mismatch en inglés) se refiere al conjunto de dificultades conceptuales y técnicas para llevar a cabo la persistencia de objetos sobre bases de datos relacionales (ver UT2).



2. Concepto de mapeo objeto-relacional

- **Mapeo – Correspondencia**
- **Planteamientos para solventar desfase O-R:**
  - **Bases de datos de objetos:** almacenan directamente objetos
  - **Bases de datos objeto-relacionales:** son bases de datos relacionales con capacidades para gestionar objetos
  - **ORM** o correspondencia/mapeo objeto-relacional.
- **ORM (Object-Relational Mapping)**
  - Consiste en el establecimiento de una correspondencia entre clases definidas en un lenguaje de programación orientado a objetos, como Java, y tablas de una base de datos relacional.
  - Proporciona mecanismos para que las modificaciones sobre los objetos se registren en la base de datos y, a la inversa, para que se pueda recuperar en objetos la información registrada en la base de datos.
  - **ODBC / JDBC:** herramientas potentes pero que necesitan un número importante de líneas de código para poder cubrir las necesidades de cada aplicación.
  - Requieren invertir tiempo en la realización de 2 modelos: relacional y orientado a objetos y además una "traducción" entre ellos.
  - Las herramientas ORM son bibliotecas o frameworks que permiten automatizar los procesos de traspaso de un sistema relacional a uno orientado a objetos.



3. HERRAMIENTAS ORM. CARACTERÍSTICAS

Ventajas

- Ayudan a reducir el tiempo de desarrollo de software
- Permiten la producción de mejor código y su reutilización
- Son **independientes de la Base de Datos**, funcionan en cualquier BD
- Proporcionan un lenguaje propio para realizar las consultas
- Incentivan la portabilidad y escalabilidad de los programas de software

Desventajas

- Las **aplicaciones son algo más lentas**, debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

Ejemplos de herramientas ORM

- Doctrine, Propel o ADOdb Active Record para incluir en proyectos PHP
- LINQ desarrollado por Microsoft para el mapeo objetorelacional para los lenguajes Visual Basic .Net y C#
- **Hibernate** desarrollado para la tecnología Java y disponible también para la tecnología .NET con el nombre de Nhibernate, es software libre bajo la licencia GNU LGPL.
- Otros: QuickDB, iPersist, Java Data Objects, Oracle, iBatis, Toplink, JPA.

4. Hibernate

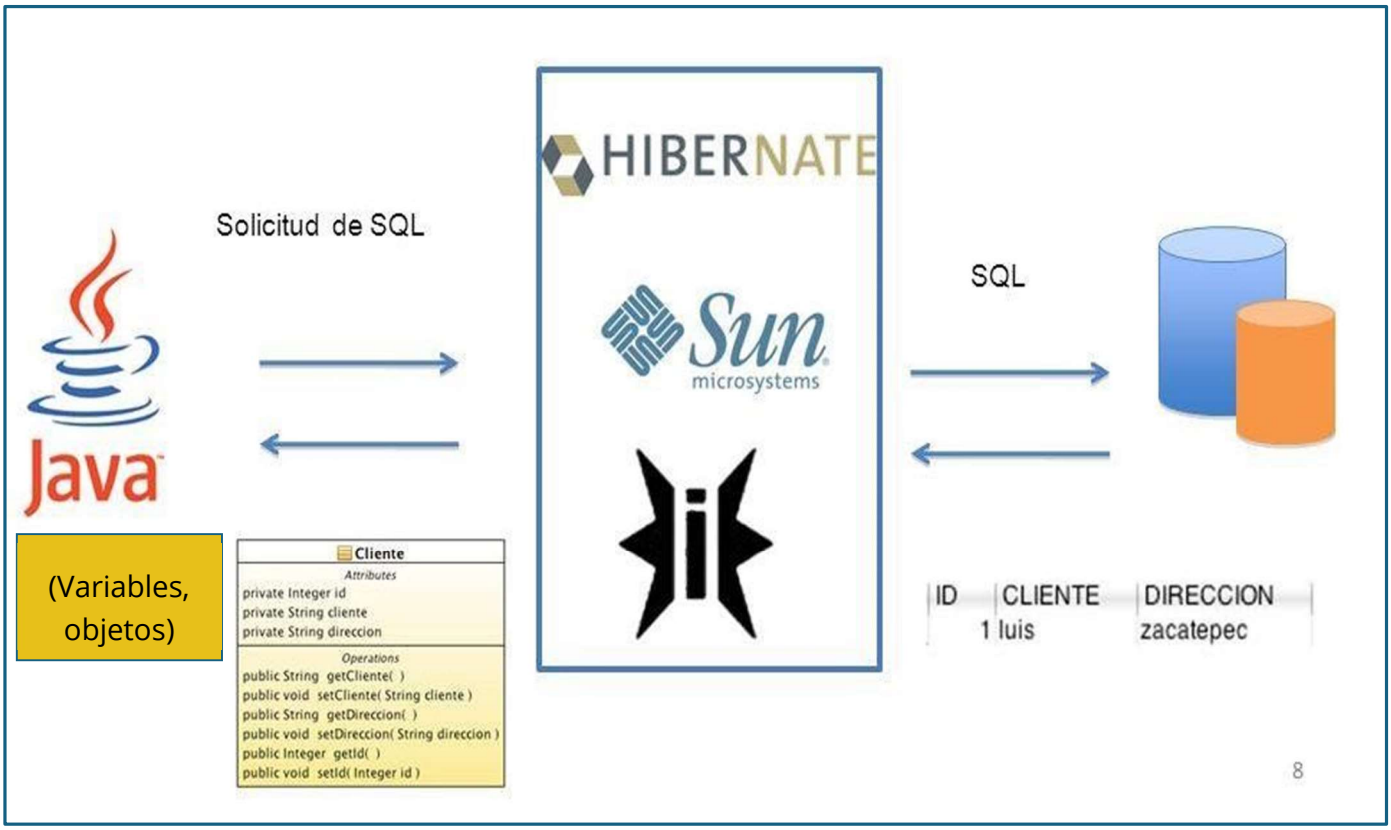
- Documentación Java sobre Hibernate:
  - Documentación sobre las distintas versiones de Hibernate incluyendo la actual: https://docs.jboss.org/hibernate/orm/
  - Documentación sobre la versión actual: https://docs.jboss.org/hibernate/orm/current/javadocs/
  - Documentación en español:

    https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/

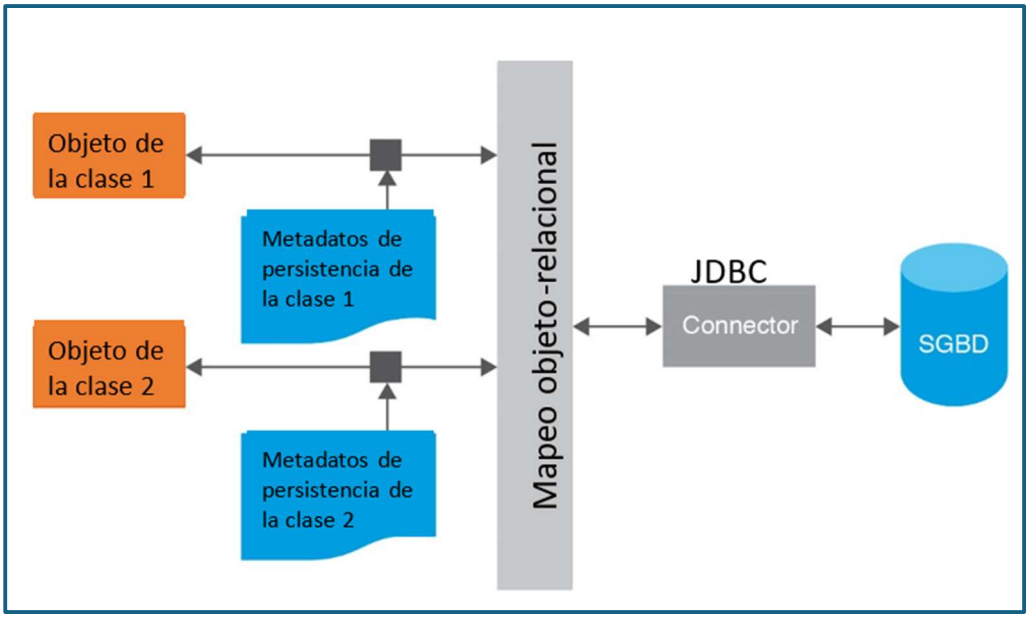
  Nota: nosotros trabajaremos con la versión **5.6.14 Final.**

- Desarrollado en el año 2001. Comenzó siendo un proyecto independiente, pero fue adquirido por JBoss en 2003.
- JBoss/Hibernate implementa las especificaciones de JPA (Java Persistence API), una parte estándar del entorno de Java EE (Enterprise Edition), la cual Sun Microsystems (ahora Oracle) introdujo como especificación para el manejo de persistencia en Java.

- **JPA** se encargó de estandarizar el mapeo entre objetos y bases de datos relacionales (un rol que Hibernate ya cubría). Por tanto, Hibernate es a menudo reconocido como una implementación líder de JPA. Hibernate se volvió parte integral de los estándares de Java EE a través de JPA.



- Facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante ficheros declarativos (XML) que permiten establecer estas relaciones.
- NO emplearemos habitualmente SQL para acceder a los datos.
- El propio motor de Hibernate construirá las consultas por nosotros.
- Dispone de Lenguaje HQL: Hibernate Query Language, que permite acceder a datos mediante Programación Orientada a Objetos.
- Las consultas HQL y SQL nativas son representadas con una instancia de *org.hibernate.Query.*



Elementos del diagrama:

- Metadatos de Persistencia:
    - Contienen la información que especifica cómo mapear los objetos de Java a las tablas de la base de datos.
    - Normalmente se definen en un **archivo de configuración** (como XML en Hibernate) o usando **anotaciones** directamente en las clases (por ejemplo, @Entity, @Table, @Column en JPA).
    - Ejemplo con anotaciones en Java: se mapea la clase Java Cliente a una tabla de base de datos llamada Clientes. Las anotaciones especifican cómo las propiedades de la clase se deben traducir a columnas de la tabla.

```
@Entity
@Table(name = "Clientes")
public class Cliente {
    @Id
    @GeneratedValue
    private int id;
    @Column(name = "nombre")
    private String nombre;
}
```

        - La clase Cliente está vinculada a la tabla Clientes.
        - La propiedad id representa la clave primaria, generada automáticamente.
        - La propiedad nombre se corresponde con la columna nombre.
        - Hibernate traducirá esta clase en una tabla con la estructura:

```
CREATE TABLE Clientes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255)
);
```

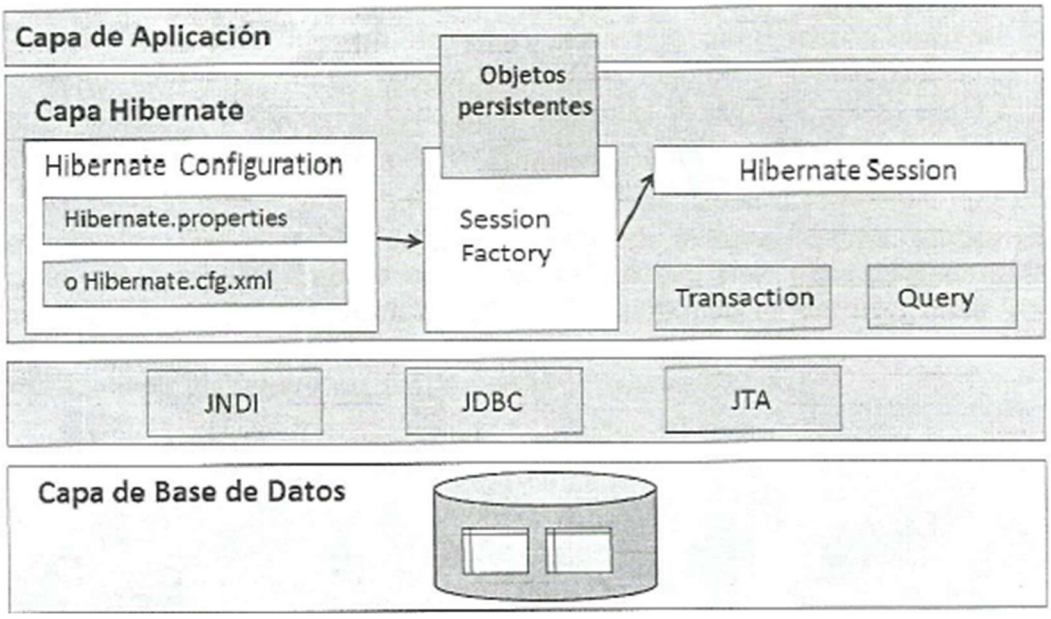
- Mapeo Objeto-Relacional:
    - Hibernate utiliza los metadatos de persistencia para transformar los objetos de Java en registros de tablas relacionales (cuando guardamos) y viceversa (cuando recuperamos).
    - Este proceso se realiza automáticamente mediante el framework, lo que reduce la necesidad de escribir código SQL manual.
- JDBC Connector (Java Database Connectivity):
    - Hibernate, internamente, utiliza JDBC para ejecutar las operaciones SQL en la base de datos.
    - JDBC es la API estándar en Java para conectarse con bases de datos. Hibernate simplifica y abstrae esta interacción, lo que permite a los desarrolladores trabajar con objetos en lugar de con consultas SQL directas.
    - Aunque Hibernate trabaja sobre JDBC, los desarrolladores no tienen que preocuparse por gestionar manualmente conexiones, sentencias SQL, o resultados.

- SGBD (Sistema de Gestión de Bases de Datos):
    - Representa la base de datos relacional donde se almacenan los datos.
    - Hibernate traduce las operaciones de Java en consultas SQL que son entendidas por el sistema de base de datos subyacente (por ejemplo, MySQL, PostgreSQL, Oracle, etc.).

4.1. Arquitectura

- Filosofía: mapear objetos Java "normales" → POJOs (Plain Old Java Objects).
- Posibles **estados de un objeto Hibernate**:
    - **Transitorio (Transient):** objeto recién instanciado mediante el operador new, y que no está asociado a una Session de Hibernate. No tiene una representación persistente en la base de datos.
    - **Persistente (Persistent):** un objeto que ya está almacenado en la base de datos. Puede haber sigo guardado o cargado, pero por definición se encuentra en el ámbito de una Session. Hibernate detecta cualquier cambio realizado a un objeto en estado persistente, y sincronizará el estado con la base de datos cuando se complete la unidad de trabajo.
    - **Separado (Detached):** un objeto está en este estado cuando cerramos al sesión mediante el método close() de Session. Una instancia separada puede ser modificada, y también puede ser asociada a una nueva sesión, haciéndola persistente de nuevo, con todas las modificaciones.

- Para almacenar y recuperar estos objetos de la base de datos se establece una conversación entre la aplicación Java y el motor de Hibernate mediante un objeto especial que es la sesión (interfaz Session):
  - Parecido al concepto de conexión de JDBC
  - Igual que con las conexiones JDBC, creamos y cerramos sesiones.
- La arquitectura de Hibernate incluye varias capas. Entre la capa de Hibernate y la base de datos se encuentran diferentes APIs Jave que usan Hibernate para interactuar con la base de datos.
- Hibernate hace uso de APIs de Java, tales como JDBC, JTA (Java Transaction Api) y JNDI (Java Naming Directory Interface).



1. Capa de Aplicación

- Es el nivel superior donde se encuentran los objetos persistentes.
- Los **objetos persistentes** son instancias de clases mapeadas por Hibernate que están asociadas a registros en la base de datos.
- Ejemplo: Una instancia de la clase Cliente podría representar un registro de la tabla Clientes.

2. Capa Hibernate

Esta capa intermedia contiene los componentes principales de Hibernate:

1. **Hibernate Configuration:**
   - Utiliza un archivo de configuración (hibernate.cfg.xml o hibernate.properties) para definir parámetros como:
     - Datos de conexión (URL, usuario, contraseña).
     - Información sobre el dialecto SQL usado por el SGBD.
     - Clases mapeadas.

2. **Session Factory:**
    - Es un componente fundamental que actúa como una **fábrica de sesiones** (Hibernate Sessions).
    - Se crea solo una vez por aplicación y maneja el ciclo de vida de las conexiones.
    - Usa la configuración para inicializarse y optimizar las interacciones con la base de datos.
3. **Hibernate Session:**
    - Representa una conexión activa entre la aplicación y la base de datos.
    - Permite realizar operaciones como crear, leer, actualizar y eliminar (CRUD) objetos persistentes.
    - Se genera a partir de la SessionFactory.
4. **Transaction:**
    - Maneja transacciones en la base de datos, asegurando que las operaciones ocurran de forma consistente y segura.
    - Ejemplo: si al guardando un objeto ocurre un error, se puede realizar un rollback con la transacción.
5. **Query:**
    - Se utiliza para realizar consultas HQL (**Hibernate Query Language**) o SQL nativo en la base de datos.
    - Ejemplo de HQL: `FROM Cliente WHERE nombre = 'Juan'`

3. Componentes de Soporte (Integración con otras capas)

- **JNDI (Java Naming and Directory Interface):**
    - Permite obtener recursos (como fuentes de datos) configurados en el servidor de aplicaciones.
- **JDBC (Java Database Connectivity):**
    - Hibernate usa JDBC como puente para comunicarse con la base de datos.
- **JTA (Java Transaction API):**
    - Se utiliza en aplicaciones avanzadas para coordinar transacciones distribuidas.

4. Capa de Base de Datos

- Es donde están los datos almacenados, en un sistema gestor de bases de datos relacional (SGBD), como MySQL, PostgreSQL, o Oracle.
- Hibernate traduce las operaciones realizadas en objetos Java en consultas SQL que la base de datos entiende.

Flujo General

1. La aplicación manipula los **objetos persistentes** en Java.
2. Hibernate utiliza los metadatos de configuración y la SessionFactory para gestionar sesiones.
3. Las sesiones realizan operaciones (mediante Transactions y Queries) sobre la base de datos.
4. Todo esto se traduce en operaciones SQL enviadas al SGBD a través de JDBC.

- Una aplicación que utiliza Hibernate trabaja con objetos tanto transitorios como persistentes y separados. Los objetos persistentes están siempre asociados a una sesión (**Session**) creada por una **SessionFactory**.
  - Una aplicación puede crear objetos transitorios y después convertirlos en persistentes para que se almacenen en la base de datos.
  - Puede también recuperar un objeto persistente desde la base de datos y realizar cambios sobre él, para que después se reflejen en la base de datos.
- Puede agrupar operaciones sobre los objetos persistentes dentro de transacciones (**Transaction**) que están siempre asociadas a una sesión.
- La aplicación puede también utilizar sentencias de HQL (Hibernate Query Language) mediante la clase **Query**.

Interfaces y clases de Hibernate

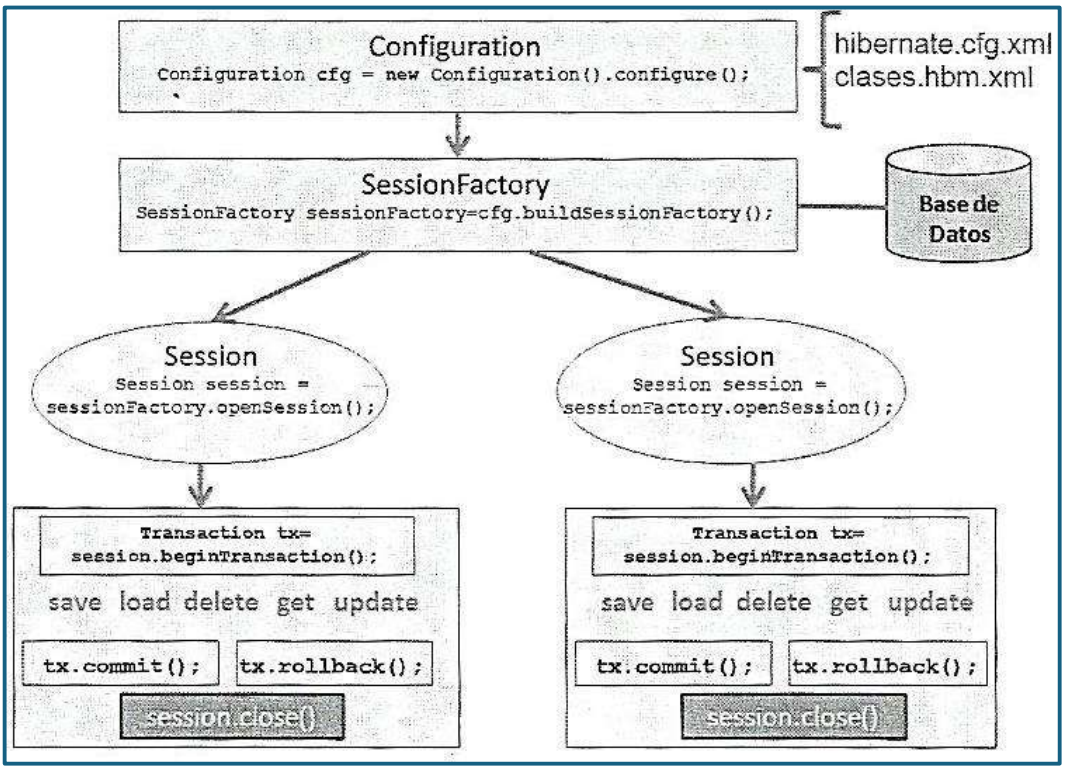
- **Interfaz Configuration (org.hibernate.Configuration)**
  - Se utiliza para configurar Hibernate.
  - La aplicación utiliza una instancia de Configuration para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate, y a continuación crea la SessionFactory.
- **Interfaz SessionFactory (org.hibernate.Session)**
  - Permite obtener instancias Session.
  - Normalmente hay una única SessionFactory para toda la aplicación, creada durante la inicialización de la misma, y se utiliza para crear todas las sesiones relacionadas con un contexto dado.
  - Si la aplicación accede a varias bases de datos se necesita una SessionFactory por cada base de datos.

- **Interfaz Session**
  - Ofrece métodos para interactuar con la BD como se hace con una conexión JDBC, pero de forma más simple:
    - ***<T> T get (Class<T> Clase, Serializable id)***: devuelve la instancia persistente de la clase indicada con el identificador dado. Devuelve *null* si la instancia no existe en la base de datos. Ejemplo: `Usuario usuario = session.get(Usuario.class, 1);`
    - ***<T> T load (Class<T> Clase, Serializable id):*** Similar a *get*, pero lanza una excepción (ObjectNotFoundException) si no encuentra la entidad.
    - ***save (Object objeto):*** guarda el objeto que se pasa como parámetro en la base de datos, convirtiendo la instancia del objeto en persistente. Devuelve el identificador generado (clave primaria) de la entidad.
    - ***persist (Object objeto):*** similar a *save*, hace persistente el objeto pero no retorna el identificador generado.
    - ***update (Object objeto):*** actualiza en la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con *load()* o *get()*.
    - ***delete (Object objeto):*** elimina de la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con *load()* o *get()*.
    - ***createQuery (String consulta):*** crea una consulta HQL (Hibernate Query Language) para obtener objetos.
    - ***close():*** cierra la conexión con la base de datos asociada a la sesión.
    - ***beginTransaction():*** inicia una nueva transacción.
    - ***commit()***: finaliza la transacción actual. Si no hay errores, se envían todas las operaciones pendientes (inserciones, actualizaciones o eliminaciones) desde la memoria a la base de datos de manera permanente.
    - ***rollback():*** deshace la transacción.

- **Interfaz Query (org.hibernate.Query)**
  - Permite realizar consultas a la base de datos y controla como se ejecutan dichas consultas.
  - Las consultas se escriben en HQL o en el dialecto SQL nativo de la base de datos que estemos utilizando.
  - Una instancia Query se utiliza para enlazar los parámetros de la consulta, limitar el número de resultados devueltos y para ejecutar dicha consulta.
- **Interfaz Transaction (org.hibernate.Transaction)**
  - Permite asegurar que cualquier error que ocurra entre el inicio y final de la transacción produzca el fallo de esta.

La siguiente figura ilustra el flujo básico para trabajar con Hibernate en una aplicación Java:

1. La configuración inicializa el entorno de Hibernate.
2. Se crea una SessionFactory basada en dicha configuración.
3. Cada operación con la base de datos se realiza a través de una Session.
4. Las transacciones garantizan la consistencia de las operaciones.
5. La sesión y la transacción se manejan para completar las operaciones y liberar recursos.



1. Configuration

- **Configuration cfg = new Configuration().configure();**
  - Este componente lee los archivos de configuración de Hibernate:
    - **hibernate.cfg.xml:** Contiene los parámetros de configuración, como las credenciales de la base de datos, el dialecto SQL, y las clases mapeadas.
    - **clases.hbm.xml:** los archivos ".hbm.xml" se utilizan para describir el mapeo entre clases Java y tablas en la base de datos (aunque con anotaciones es menos común en proyectos modernos).
  - A partir de esta configuración, se inicializa Hibernate.

2. SessionFactory

- **SessionFactory sessionFactory = cfg.buildSessionFactory();**
  - La SessionFactory es una **fábrica de sesiones**, creada una sola vez por la aplicación.
  - Proporciona conexiones (sessions) a la base de datos y está optimizada para manejar varios usuarios o peticiones concurrentes.
  - Internamente, también administra la conexión con la base de datos.

3. Session

- **Session session = sessionFactory.openSession();**
    - Cada operación con Hibernate comienza con la creación de una sesión.
    - La **Session** representa una conexión con la base de datos y es usada para realizar todas las operaciones como insertar, consultar, actualizar o eliminar.

4. Transaction

- **Transaction tx = session.beginTransaction();**
    - Una transacción envuelve un conjunto de operaciones de la base de datos, garantizando que se ejecuten completamente o se deshagan en caso de error.
    - Operaciones principales dentro de una transacción:
        - **save:** Insertar un nuevo objeto en la base de datos.
        - **load/get:** Recuperar un objeto desde la base de datos.
        - **delete:** Eliminar un objeto.
        - **update:** Actualizar los datos de un objeto existente.
    - Finalización de la transacción:
        - **tx.commit();**: Confirma la transacción y aplica los cambios.
        - **tx.rollback();**: Revierte los cambios en caso de fallo.

5. Cierre de sesión

- **session.close();**
    - Es importante cerrar la sesión después de usarla para liberar los recursos utilizados (memoria, conexiones, etc.).

4.2. Estructura de ficheros

1. Archivo de configuración: **hibernate.cfg.xml**. Contiene todo lo necesario para realizar la conexión a la base de datos.
2. Archivo de ingeniería inversa: **hibernate.reveng.xml**. Encargado de crear las clases asociadas a las tablas de la base de datos.
3. Clases Java Hibernate y ficheros de mapeo:
    - Archivos .java que contienen las clases Java “equivalentes” a las tablas de la base de datos (POJOs, Plain Old Java Objects): **<nombre_clase>.java**
    - Ficheros de mapeo:**<nombre_clase>.hbm.xml**). Contiene la información del mapeo de cada clase a su respectiva tabla de la base de datos.
4. Fichero **HibernateUtil.java**: clase que facilita la inicialización, configuración y gestión de la sesión de Hibernate.