

*Soporte a DAO
(DAO Support)*



¿Por qué DAO?

El soporte de Objeto de Acceso a Datos en Spring (en inglés Data Access Object o DAO) está dirigido a proveer consistencia y portabilidad al código, sin importar qué tecnologías de acceso a datos (persistencia) estemos utilizando, ya sea JDBC, Hibernate, JPA o JDO

El cambio de una tecnología de persistencia a otra es una cuestión de cambiar unas pocas líneas de código en las anotaciones (inyección de dependencia)

También permite a nuestro código no preocuparse por la captura de excepciones, que son específicas para cada tecnología de persistencia



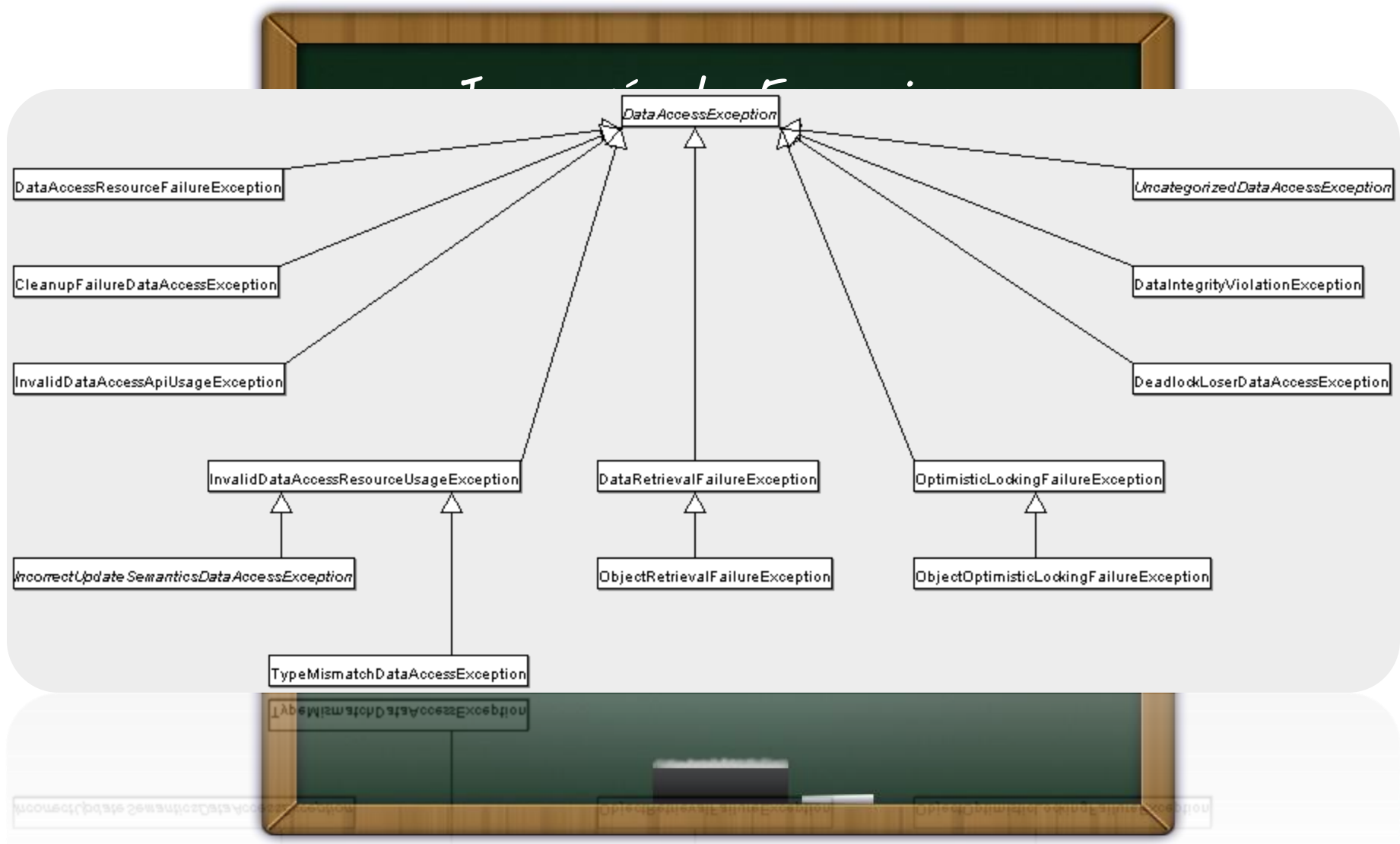
Jerarquía de Excepciones

Spring proporciona muy buena traducción de excepciones de las tecnologías de persistencia como SQLException

Spring maneja su propia jerarquía de clases de excepciones con el DataAccessException como la clase padre

Estas excepciones envuelven la excepción original, de forma simple y en un lenguaje claro, por lo tanto no se corre el riesgo de que se vaya a perder información sobre lo que pudo haber salido mal

Maneja tanto excepciones de JDBC y de JPA / Hibernate



*Anotación
@Repository*



Anotación @Repository

Garantiza que nuestros DAOs o repositorios tengan una adecuada traducción de errores/excepciones

Permite el soporte para el escáner de componentes DAOs/Repositorios y que estos puedan ser encontrados y configurados sin tener que ser definidos en el XML de Contexto de Spring

@Repository es un estereotipo de @Component



Anotación @Repository

@Repository

```
public class AlgunDao implements IDao {
```

```
// ...
```

```
}
```



Acceso a recursos de persistencia

Cualquier implementación de DAO o repositorio tiene que acceder a un recurso de persistencia, dependiendo de la tecnología de persistencia usada:

- Dao basado en JDBC necesita acceder al recurso DataSource JDBC
- Dao basado en Hibernate necesita acceder al SessionFactory
- Dao JPA necesita acceder al EntityManager
- DAO JPA usando API Spring Data, la interface CrudRepository

La forma más simple de acceder a estos recursos es usar inyección de dependencia para inyectarlos

- @Autowired para inyectar un DataSource JDBC o Hibernate SessionFactory
- @PersistenceContext para inyectar un EntityManager de JPA

Soporte ORM en Spring



Soporte ORM en Spring

Spring soporta administración de recursos, implementación data access object (DAO) y control de transacción

- Hibernate
- JPA (Java Persistence API)
- JDO
- iBATIS SQL Maps

Beneficios de usar DAOs de Spring

Fácil de testear

Excepciones de acceso a datos más entendibles

Mejor administración de recursos

Fácil configuración del JPA e Hibernate

Beneficios de usar DAOs de Spring

Integrado con transaction management
(manejo de transacciones)

- Podemos envolver nuestro código ORM (DAOs) con una declarativa programación orientada a aspectos (AOP) ya sea anotando un método con `@Transactional` el cual será interceptado por el aspecto para aplicar la transacción



Porqué el Mapeo de Objeto Relacional (ORM)?



- La capa de persistencia es una importante parte de cualquier proyecto de desarrollo de aplicaciones empresariales
 - ✓ Accede y opera con los datos persistentes típicamente de una base de datos relacional
- Las Bases de datos relacionales manejan tablas (con filas y columnas)
 - Nosotros, los desarrolladores Java, queremos trabajar con clases/objetos, no filas y columnas
 - ORM se encarga del mapeo entre los dos



Porqué el Mapeo de Objeto Relacional (ORM)?

ORM maneja relaciones de objetos



Diseñado con un excelente rendimiento, rápidas consultas/operaciones en la BBDD, maneja dos niveles de cache, donde el primer nivel es automático





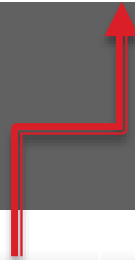
El mapeo objeto-relacional (Object-Relational mapping, o sus siglas ORM) es una arquitectura que permite trabajar con los datos de una base de datos relacional (RDBMS) en forma de objetos (lenguaje POO)

Esto crea una base de datos orientada a objetos en nuestra aplicación mapeada a una base de datos relacional, es decir una base de datos virtual de objetos, sobre la base de datos relacional



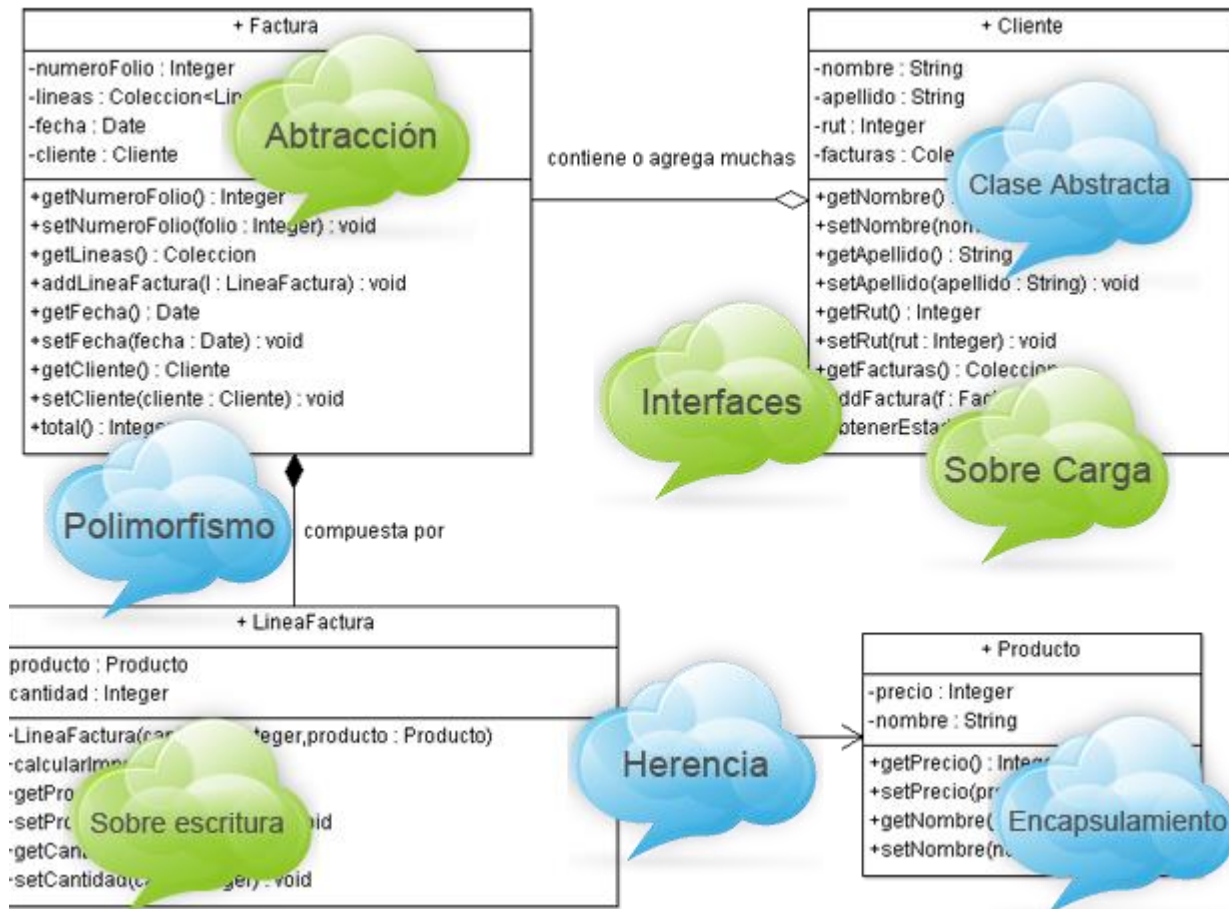
- ✓ *Esto funciona asociando a cada tabla de la base de datos un Plain Old Java Object (POJO o clase Entity)*
- ✓ *Cada atributo de la clase Entity es mapeado o asignado a las columnas de la tabla de la base de datos*
- ✓ *Un Entity es similar a una Java Bean, con propiedades accesibles mediante métodos setter y getter*

id	cliente_id	solicitud_id	status_id	created_at	updated_at	numero_operacion	plazo_entrega	total	descripcion
1	6	1	4	2014-02-16 20:38:32	2014-02-16 20:38:55	VE10001	2014-02-16 20:38:55	8889	alguna nota

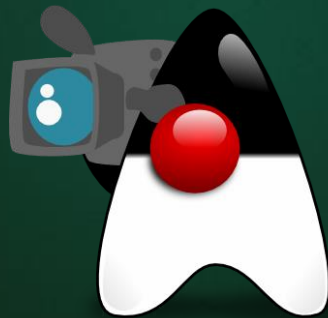


id	factura_id	producto_id	cantidad	importe
2	1	1	4	8888

Esto permite el uso de las características propias de la programación orientada a objetos (relaciones, herencia y polimorfismo)

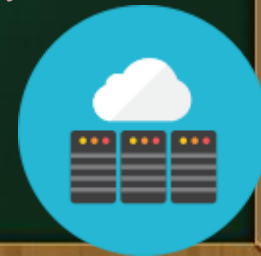


Clases de Dominio o Entidades



Clases de Dominio o Entity

- *Las clases de dominio son aquellas en una aplicación que implementan las entidades del dominio de negocio (por ejemplo, Cliente y OrdenCompra en una aplicación de E-commerce)*
- *Un ORM como Hibernate o JPA funciona mejor si estas clases siguen algunas reglas simples, también conocido como el modelo de programación Plain Old Java Object (POJO).*



Clases de Dominio o Entity

Objetos
Mapeados

Objetos
Manejados

Objetos
Anotados

Objetos
Persistentes

Objetos
De Consultas

Objetos
Cacheados

Los objetos son instancias
que solo viven en
memoria

Los entity son objetos que
viven en modo persistente
en la Base de Datos



Una entidad es un objeto de persistencia que representa una tabla en el modelo de datos relacional y cada instancia de esta entidad corresponde a un registro en esa tabla

El estado de persistencia de una entidad se representa a través de atributos persistentes



Estos atributos usan Anotaciones para el mapeo de estos objetos en el modelo de base de datos

Usamos Anotaciones de JPA para mapear las clases de dominio de persistencia

Pasos para escribir una clase Entity

Paso 1: Implementar constructor sin argumentos

- Toda clase persistente deben tener un constructor por defecto de manera que JPA pueda crear instancias de ellas

Paso 2: Proveer un atributo Id (identificador)

- Este atributo se asigna (mapeada) al campo de llave primaria de una tabla de base de datos.
- Este atributo puede ser nombrado como se quiera, y el tipo de dato puede ser cualquier primitivo `java.lang.String` o `java.lang.Integer` o `java.lang.Long`

Pasos para escribir una clase Entity

Paso 3: Declarar los métodos getter/setter para los atributos persistentes

Paso 4: Implementar la interfaz `java.io.Serializable`





¿Qué es Hibernate?

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java, es el Framework más popular de Persistencia que habilita la persistencia en clases POJO de forma transparente.

Hibernate es el proveedor por defecto de JPA en Spring Boot, es decir, JPA provee una especificación abstracta de ORM en Java EE, como un estándar e Hibernate es un proveedor que implementa dicha especificación.

Facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante objetos persistentes que siguen fielmente los conceptos y paradigmas comunes de la programación orientada a objetos

Utiliza anotaciones en las clases de las entidades que permiten establecer estas relaciones.



Mapeo Anotaciones



¿Cómo hace JPA/Hibernate para mapear Usando anotaciones?

JPA asigna objetos a una base de datos mediante el uso de metadatos

Las Entidades tienen asociados metadatos que describen el mapeado

Estos metadatos son anotaciones descritas en `javax.persistence`

JPA cuenta con reglas de mapeados por defecto, solo necesitamos anotaciones para excepciones, configuración por excepciones

Mapeo de una clase Entidad

```
@Entity
@Table(name = "estudiantes")
public class Estudiante {
```

Para ser reconocida como una Entidad la clase debe aparecer con la anotación **@Entity**

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
```

La anotación **@Id** denota la llave primaria, su valor es generado por **@GeneratedValue**

```
    @Column(length = 50, nullable = false)
    private String nombre;
```

```
    @Column(name = "fecha_nacimiento")
    private Date fechaNacimiento;
```

@Column usado en atributos para sobrescribir el mapeado de columnas por defecto

```
    private int nota;
```

```
    public Estudiante() {
    }
```

```
    public Estudiante(String nombre, Date fechaNacimiento, int nota){
        this.nombre = nombre;
        this.fechaNacimiento = fechaNacimiento;
        this.nota = nota;
    }
```

```
    ...etc... Getters y setters
```

Asigna Entity Estudiante a la tabla estudiantes

Genera una clave primaria incremental

Sincroniza valores de atributos en columnas de la BD

Mapeo de una clase Entidad

- *@Entity*: Indica que es una clase POJO de Entidad (representa una tabla en la base de datos)
- *@Table*: Especifica la tabla principal relacionada con la entidad.
 - *name* - nombre de la tabla, por defecto el de la entidad si no se especifica
 - *catalog* - nombre del catálogo
 - *schema* - nombre del esquema

Mapeo de una clase Entidad

- *@Id*: Indica la clave primaria de la tabla
- *@GeneratedValue*: Asociado con la clave primaria, indica que ésta se debe generar por ejemplo con una secuencia de la base de datos
 - *strategy* - estrategia a seguir para la generación de la clave: *AUTO* (valor por defecto, el contenedor decide la estrategia en función de la base de datos), *IDENTITY* (utiliza un contador, ej: *MySQL*), *SEQUENCE* (utiliza una secuencia, ej: *Oracle*, *PostgreSQL*) y *TABLE* (utiliza una tabla de identificadores).
 - *generator* - forma en la que genera la clave.

Mapeo de una clase Entidad

- *@Column: Especifica una columna de la tabla a mapear con un campo de la entidad*
 - *name - nombre de la columna.*
 - *unique - si el campo es único*
 - *nullable - si permite nulos.*
 - *insertable - si la columna se incluirá en la sentencia INSERT*
 - *updatable - si la columna se incluirá en la sentencia UPDATE*
 - *length - longitud de la columna.*
 - *precision - número de decimales*
 - *scale - escala decimal*

*¿Qué es un DAO?
Data Access Object*



¿Qué es un DAO?

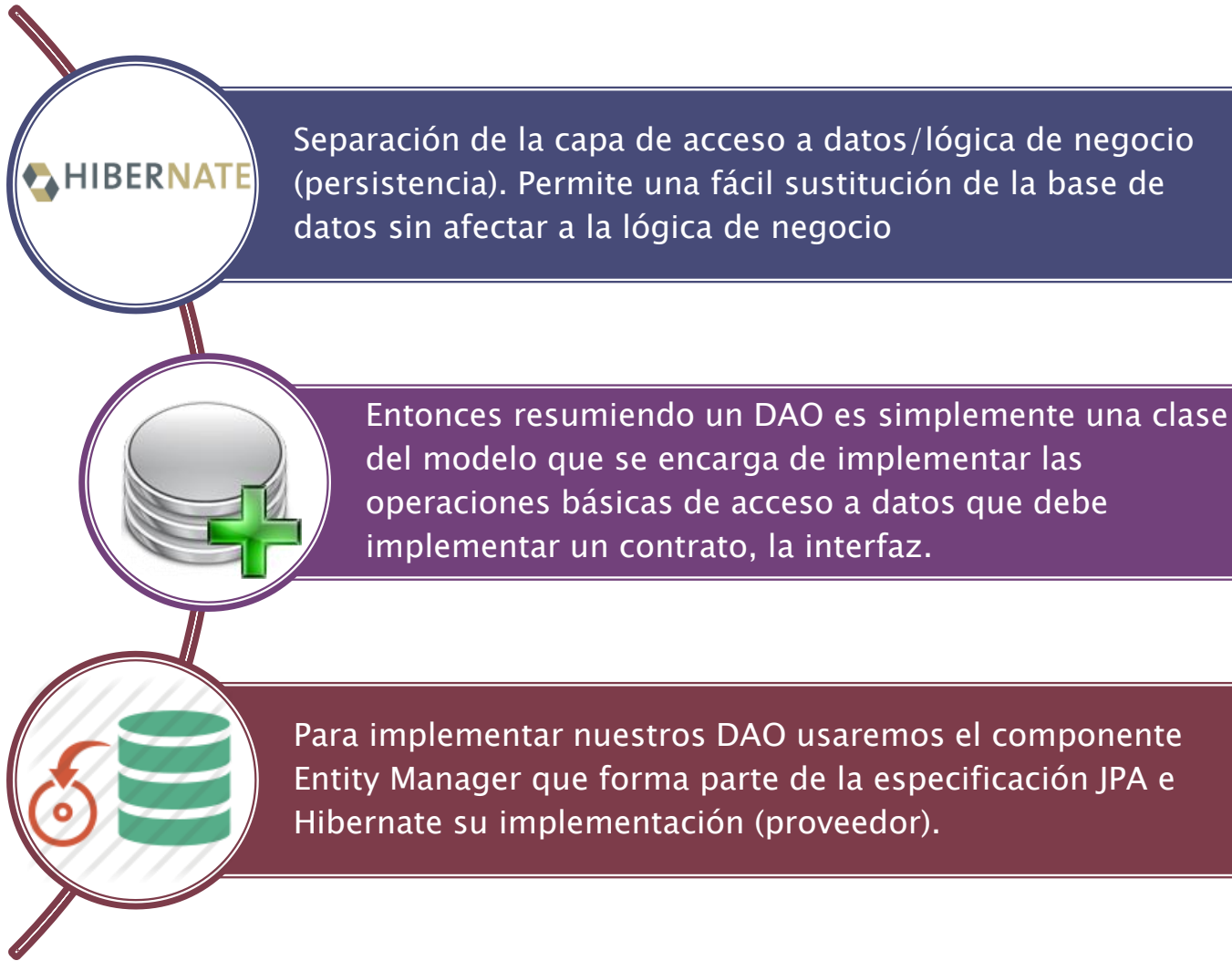


En Ingeniería de software, un Data Access Object (DAO) es una clase que implementa y provee una interfaz común para acceder y trabajar con los datos, independiente de las tecnologías a utilizar JDBC, JPA, Hibernate, TopLink, OpenJpa, Codo, EclipseLink, iBATIS o JDO

Esta interface tiene que tener los métodos necesarios para recuperar y almacenar los datos (contrato de implementación) con las operaciones básicas: listar, obtener por id, guardar, eliminar etc




¿Qué es un DAO?




Data Access Object (DAO)



Data Access Object (DAO)



También se encarga de realizar y ejecuta las consultas del tipo SELECT (ORM HQL– Hibernate Query Language): crear el objeto, prepara las consultas y ejecutar



*Especificación
JSR-338
JPA*



Java Persistence API, más conocida por su sigla JPA, es la API de persistencia desarrollada para la plataforma Java EE e incluida en el estándar EJB3

Busca unificar las diferentes tecnologías que proveen un mapeo objeto-relacional bajo un único estándar

El objetivo del diseño de esta API es no perder de vista las ventajas de la orientación a objetos al interactuar con una base de datos

y usar objetos simples (conocidos como Entity o POJOs).



Java Persistence API

Para trabajar con la especificación JPA en spring, primero tenemos que seleccionar un proveedor o implementación de JPA

Hibernate es la implementación preferida

Entre las posibles opciones tenemos

JPA



Hibernate

OpenJPA

EclipseLink

Spring MVC

Capa de Negocio

ORM Vendor
ej. Hibernate

Capa Persistencia – JPA



Seleccionar un proveedor JPA

En JPA, debemos seleccionar un proveedor o implementación en la configuración de contexto de spring



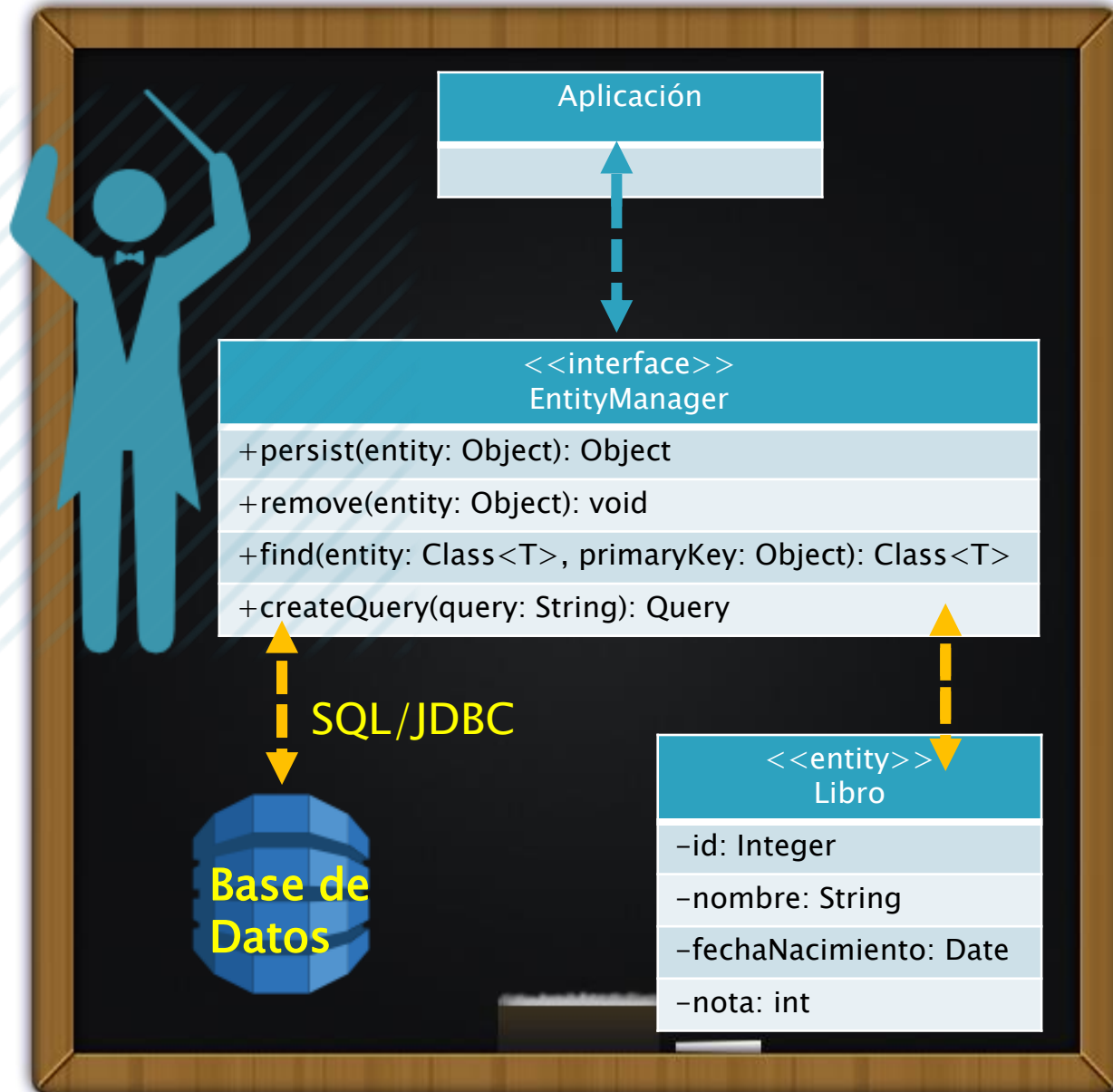
Por ejemplo, podemos escoger entre los siguientes proveedores JPA

- Hibernate
- OpenJPA
- EclipseLink

JPA Entity Manager



Clase Entity y el EntityManager



Es la interfaz en la que se apoya la API de persistencia y la que se encarga del mapeo entre una tabla relacional y su objeto de entidad Java (Entity)

Proporciona métodos para manejar la persistencia de un objeto de Entidad, permite agregar, eliminar, editar, consultar (JPQL) y manejar su ciclo de vida

En general se encarga de manejar o administrar el contexto de persistencia y sus entidades

Es el director de orquestas de las entidades JPA



JPA - Entity Manager

- Sus métodos más importantes son
 - `persist(Object entity)`: Almacena un objeto entity en el contexto de persistencia y en la base de datos
 - `merge(Object entity)`: Actualiza las modificaciones en una entidad devolviendo un objeto entity manejado por el contexto JPA
 - `remove(Object entity)`: Elimina la entidad
 - `find(Class<T> entity, Object primaryKey)`: Busca la entidad a través de su clave primaria



JPA - Entity Manager

- *flush(): Sincroniza las entidades con la base de datos, hace el comit*
- *createQuery(String query): Crea una query utilizando el lenguaje JPQL o HQL*
- *createNativeQuery(): Crea una query utilizando el lenguaje SQL*



Acceso al contexto de Persistencia

- Cualquier implementación DAO o Repositorio en Spring tendrán que acceder a un contexto de persistencia JPA
- Necesita acceder al `EntityManager`, por lo tanto es necesario Inyectar la referencia
- Podemos obtener una referencia al `EntityManager` a través de la anotación `@PersistenceContext`.

```
@PersistenceContext  
private EntityManager entityManager;
```

*Implementar DAO
con JPA
EntityManager*



Implementación DAO usando JPA

@Repository("estudianteDao")

public class JpaEstudianteDao implements EstudianteDao {

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void save(Estudiante estudiante) {
 entityManager.merge(estudiante);
}

@Transactional

public void delete(Integer estudianteId) {
 Estudiante estudiante = entityManager.find(Estudiante.class, estudianteId);
 entityManager.remove(estudiante);
}

@Transactional(readOnly = true)

public Estudiante findById(Integer estudianteId) {
 return entityManager.find(Estudiante.class, estudianteId);
}

@Transactional(readOnly = true)

public List<Estudiante> findAll() {
 Query query = entityManager.createQuery("from Estudiante");
 return query.getResultList();
}
}

Implementación DAO usando JPA

El DAO no tiene ninguna dependencia directa de Spring (salvo las transacciones), sin embargo se integra muy con Spring

Además, el DAO utiliza la anotaciones `@PersistenceContext` (propia de Java EE) para requerir la inyección del `EntityManager`