0ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 4

Hibernate Extras

Ing. Jorge Alberto Montalvo Olvera

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

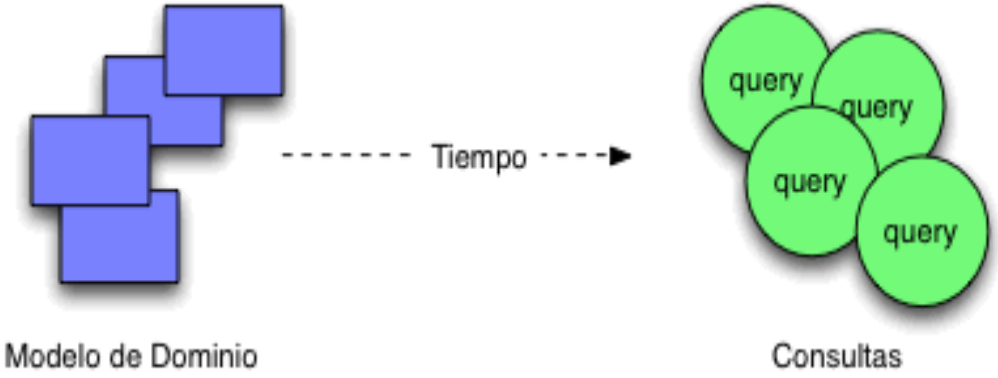
Educación Continua
1971 - 2021

TEMARIO

- Named Querys
- Carga Lazy
- Modelo DAO
- Caché

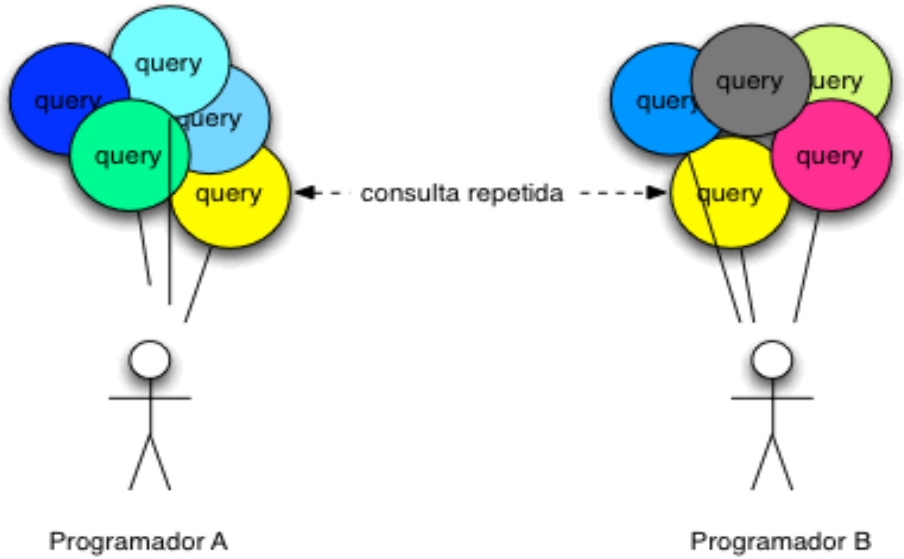
Named Querys

- Al realizar el desarrollo de un proyecto una vez que se tiene definido el modelo de dominio el otro gran trabajo importante es construir las diferentes consultas a realizar contra el modelo.



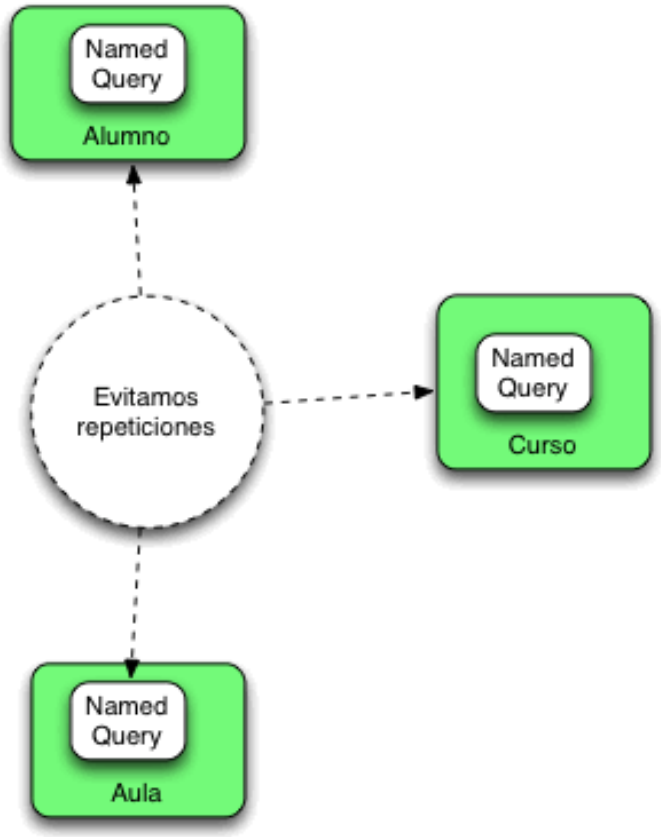
Named Querys

- Uno de los problemas que nos podemos encontrar cuando construimos las consultas es que varios desarrolladores van a trabajar en el mismo proyecto construyendo consultas. Con lo cual puede que acabemos con consultas repetidas en distintas partes del código .



Named Querys

- Para evitar este problema podemos apoyarnos en NamedQueries que nos permiten definir las consultas a nivel de clase de dominio evitando las repeticiones.



Named Querys

- @NamedQuery
  - Las consultas con nombre se definen junto con la entidad, utilizando la anotación @NamedQuery. Mejoran el rendimiento con respecto a las consultas dinámicas, ya que son procesadas una única vez

```
@NamedQuery(name="salarioPorNombreDepartamento",
                  query="SELECT e.salary " +
                        "FROM Empleado e " +
                        "WHERE e.departamento.nombre = : deptNombre
AND " +
                        "      e.nombre = : empNombre")
```

Named Querys

- @NamedQueries
  - Si se necesita más de una consulta para una entidad, deben incluirse en la anotación @NamedQueries, que acepta una array de una o más anotaciones @NamedQuery

```
@NamedQueries({
            @NamedQuery(name="Empleado.findAll",
                        query="SELECT e FROM Empleado e"),
            @NamedQuery(name="Empleado.findById",
                        query="SELECT e FROM Empleado e WHERE e.id =
:id"),
            @NamedQuery(name="Empleado.findByNombre",
                        query="SELECT e FROM Empleado e WHERE
e.nombre = :nombre")
        })
```

Named Querys

- Para ejecutar la consulta hay que llamar al método createNamedQuery pasándole como parámetro el nombre de la consulta.

```
//Hibernate Named Query
    TypedQuery query = session.getNamedQuery("findEmployeeByName");
    query.setParameter("name","amit");

    List<Employee> employees=query.getResultList();
```

Carga Lazy

- Uno de los puntos más importantes a analizar en un software es el desempeño y optimizar al máximo las queries SQL, esto puede resultar en una ganancia de desempeño considerable.

- Un uso incorrecto provoca una gran pérdida en el desempeño de una aplicación, lo que obliga a la base de datos a querys innecesarias.

Carga Lazy

- Cuando modelamos un sistema usando la orientación a objetos, creamos varias relaciones que pueden ser @OneToOne, @ManyToOne, @OneToMany o @ManyToMany y para la base de datos, cada relación es una nueva tabla a consultar.

Carga Lazy

- Tipos de Inicialización de Propiedades
  - FetchType.LAZY = Esto no carga las relaciones a menos que lo invoque a través del método getter.
    - Lazy initialization mejora el rendimiento al evitar cálculos innecesarios y reducir los requisitos de memoria.
  - FetchType.EAGER = Esto carga todas las relaciones.
    - Eager initialization consume más memoria y la velocidad de procesamiento es lenta.

Carga Lazy

- Por estándar, cuando la relación se anota con @OneToOne o @ManyToOne, se carga en modo Eager

```
@Entity
public class Alumno {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    @OneToOne
    private Matricula matricula;
```

Carga Lazy

- Por estándar, cuando la relación está anotada con @OneToMany o @ManyToMany, se carga en modo Lazy

```
@Entity
public class Alumno {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    @OneToMany
    private List<Matricula> matriculas;
```

Carga Lazy

- Por el modo Lazy, solo cargamos informaciones cuando ejecutamos un getter para hacer un alumno.getXXX().getYYY().
  - Para obtener el dato YYY de todas las XXX del alumno puede traer problema de desempeño a la aplicación, ya que se ejecutarán varias queries.

```
for(Alumno alumno : alumnos) {
    for(Matricula matricula : alumno.getMatriculas(){
          System.out.println(matricula.getCodigo());
    }
}
```

Carga Lazy

- Este problema se conoce como consultas [N + 1]

- Además del problema mencionado anteriormente, debemos tener cuidado con LazyInitializationException.

Carga Lazy

- La mejor manera de resolver la LazyInitializationException es usar la directiva JOIN FETCH en las consultas de su entidad.

```
SELECT e
   FROM Empleado e JOIN FETCH e.direccion
```

- @Fetch
  - Con la anotación @Fetech la recuperación de la base de datos se realiza por JOIN fetch
  - @FetchMode.JOIN

```
@OneToMany(mappedBy="tableName", cascade=CascadeType.ALL)
@Column(name="id")
@Fetch(FetchMode.JOIN)
```

DAO

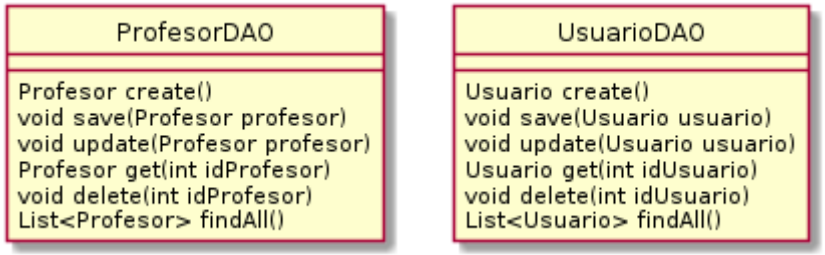
- El patrón Data Access Object (DAO) pretende principalmente independizar la aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos.

- Centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO.

- Fuera de las clases DAO no debe haber ningún tipo de código que acceda al repositorio de datos.

DAO

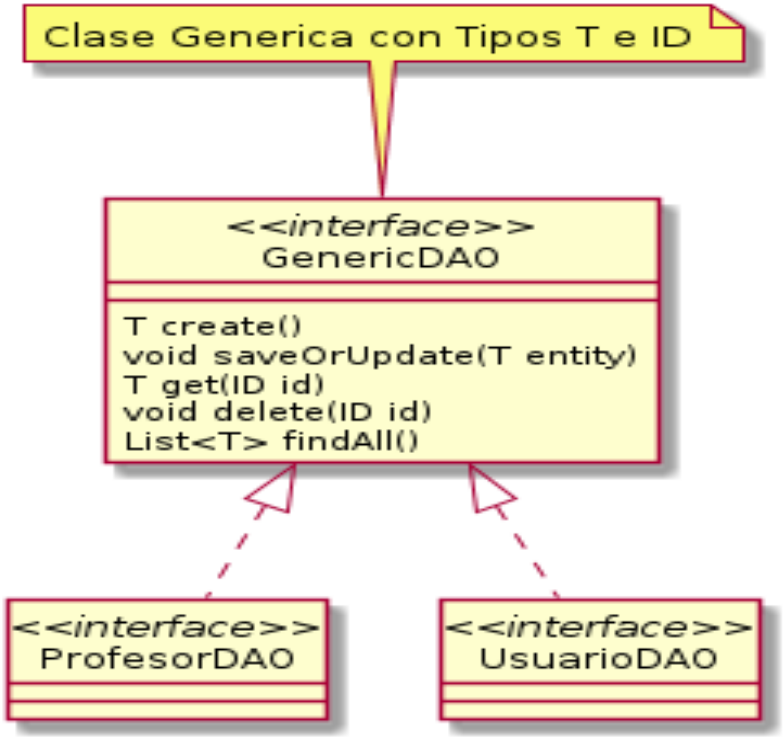
- Las ventajas de usar el patrón DAO son las siguientes:
  - Modificar el API de acceso
  - Modificar el repositorio de datos



- El mayor problema del patrón DAO es la repetición de código para cada una de las entidades.

DAO

- La solución es crear una interface genérica DAO de la cuál saldrán varias interfaces DAO hijas y lo que permitirá varias implementaciones.

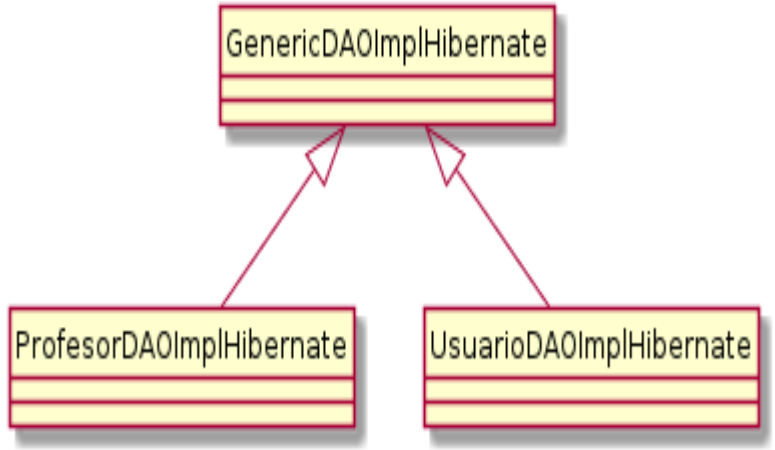


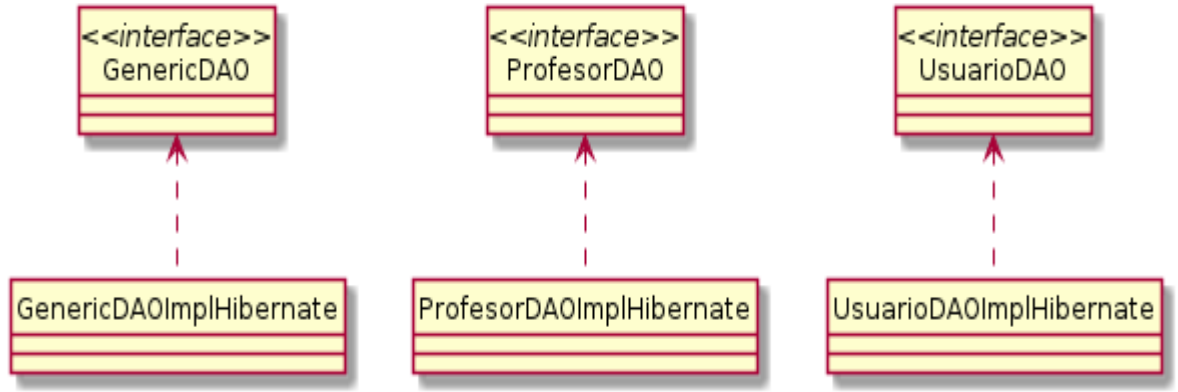
DAO

- La definición de los métodos quedaría:
  - T create(): Crea un nuevo objeto de la entidad.
  - void saveOrUpdate(T entity) : Inserta o actualiza un objeto de una entidad en la base de datos.
  - T get(ID id) : Obtiene un objeto de una entidad desde la base de datos en función de la clave primaria.
  - void delete(ID id) : Borra un objeto de una entidad de la base de datos en función de la clave primaria.
  - List<T> findAll() : Obtiene una lista con todos los objetos de una entidad de la base de datos.

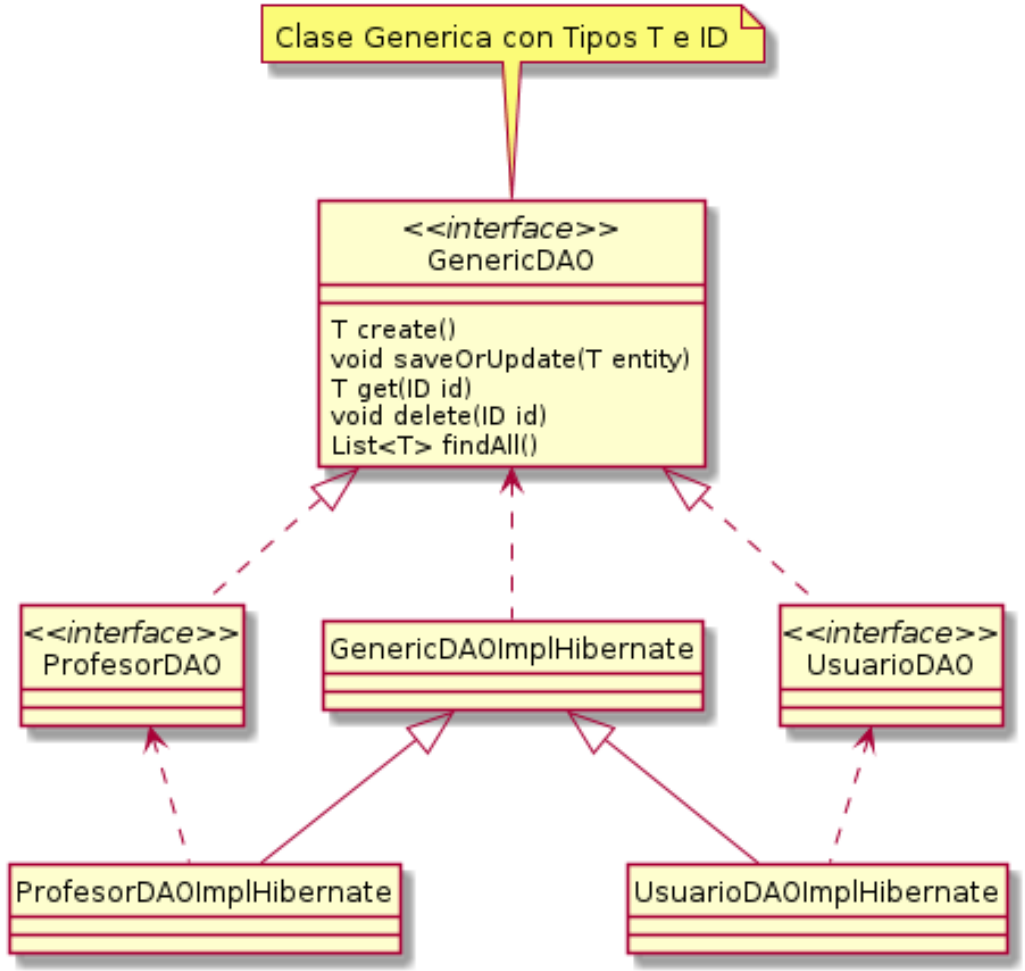
DAO

- La implementación de este nuevo modelo DAO quedaría:





DAO


Clase Generica con Tipos T e ID
<<interface>>
GenericDAO
T create()
void saveOrUpdate(T entity)
T get(ID id)
void delete(ID id)
List<T> findAll()
<<interface>>
ProfesorDAO
GenericDAOImplHibernate
<<interface>>
UsuarioDAO
ProfesorDAOImplHibernate
UsuarioDAOImplHibernate

Caché

- El almacenamiento en caché es un mecanismo para almacenar copias de datos o archivos de tal manera que se puedan servir rápidamente. El almacenamiento en caché está relacionado con un componente de hardware o software que almacena datos para que las solicitudes futuras de esos datos puedan ser atendidas más rápidamente.

Caché

- Si estamos hablando de caché de base de datos, el almacenamiento en caché actuará como una memoria intermedia que permanece entre la aplicación y la base de datos.

- El almacenamiento en caché de Hibernate actúa como una capa entre la base de datos real y su aplicación. Reduce el tiempo necesario para obtener los datos necesarios, ya que se recuperan de la memoria en lugar de acceder directamente a la base de datos. Es muy útil cuando necesita obtener el mismo tipo de datos varias veces.

Caché

- Existen principalmente dos tipos de almacenamiento en caché:
  - Caché de primer nivel
  - Caché de segundo nivel

- La caché de primer nivel está habilitada por defecto por Hibernate.

- El objeto de sesión mantiene la caché de primer nivel.

Caché

- Una aplicación puede tener muchas sesiones.

- Los datos retenidos por un objeto de sesión no son accesibles para toda la aplicación (los datos de una sesión en particular no se comparten con otras sesiones de la aplicación).

Caché

- Hibernate es una herramienta ORM que se utiliza para simplificar las operaciones de la base de datos. “Convierte objetos convertidos en relaciones (para almacenar en DB) y viceversa”.

- Entonces, cuando consulta una entidad u objeto, por primera vez se recupera de la base de datos y se almacena en el caché de primer nivel (asociado con la sesión de hibernate). Si volvemos a consultar la misma entidad u objeto con el mismo objeto de sesión, se cargará desde la caché y no se ejecutará ninguna consulta SQL.

Caché

- Algunos métodos útiles:
  - Session.evict (): para eliminar la entidad almacenada en caché.
  - Session.refresh (): método para actualizar la caché.
  - Session.clear (): método para eliminar todas las entidades del caché.

Contacto

Jorge Alberto Montalvo Olvera

Ingeniero en Computación

jorge.amontalvoo@gmail.com