ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 Materia: **Java Web**

TEMA: JPA

Agenda

- Introducción a JPA
- Entidades
- Annotations
- Clases Embeddable
- Claves Primarias Compuestas
- Estrategias de Herencia
- Unidad de Persistencia
- Gestor de Entidades
- Operaciones con los Datos

Introducción a JPA

- JPA es la especificación de un API de persistencia desarrollada para las plataformas JavaEE y JavaSE.
- Busca unificar la manera en que funcionan las utilidades que proveen mapeo objeto relacional (ORM).
- Existen varias implementaciones como *Hibernate*, *EclipseLink*, etc.
- Permite mantener la persistencia en bases de datos de forma tal de librar al desarrollador de dicha tarea.
- Sus componentes se encuentran en el paquete *javax.persistence*
- Provee un lenguaje de consultas independiente de la plataforma: JPQL (Java Persistence Query Language)
- También define un conjunto de ***annotations*** para incorporar metadatos a las clases que representan las entidades de persistencia.

Entidades (1)

- Una entidad es un objeto de dominio de persistencia que normalmente representa una tabla en el modelo de datos relacional, y cada instancia de esa entidad a un registro de esa tabla.
- Para realizar los mapeos de las entidades a la base de datos existen dos alternativas:
  - Utilizar un archivo xml (por ejemplo orm.xml).
  - Utilizar annotations en las propias clases de las entidades (clases POJO, Plain Old Java Objects, con constructor sin parámetros y métodos get (o is) y set para acceder a sus propiedades.

Entidades (2)

- Ejemplo de mapeo en archivo xml

```
<entity-mappings ...>
  <entity class="entidades.Usuario">
    <table name="USUARIOS" />
    <attributes>
      <id name="nick" />
      <basic name="pass">
        <column name="PASSWORD" optional="false" />
      </basic>
    </attributes>
  </entity>
</entity-mappings>
```

Entidades (3)

- Ejemplo de mapeo en archivo annotations

```
import javax.persistence.*;
...
@Entity
@Table(name="USUARIOS")
public class Usuario {
    @Id
    private String nick;
    @Column(name="PASSWORD", nullable=false)
    private String pass;
    public Usuario() { }
    // Getters y Setters...
}
```

Annotations (1)

- Básicas:
  - *@Entity*: Declara una clase como una entidad de persistencia.
  - *@Table*: Permite indicar el nombre de la tabla a la que mapea una entidad.
  - *@Id*: Especifica la propiedad de la entidad que será utilizada como identificador (clave primaria en la tabla).
  - *@GeneratedValue*: Especifica la estrategia (strategy) de autogeneración del valor del identificador de la entidad (por ejemplo GenerationType.AUTO).
  - *@Transient*: Indica que la propiedad no será persistida en la tabla.
  - *@Column*: Permite indicar el nombre de la columna a la que mapea una entidad, y otros atributos de la misma (como length, nullable, unique, etc.).

Annotations (2)

- Mapeo Relaciones:
  - *@Entity*: Declara una clase como una entidad de persistencia.
  - *@ManyToMany*: Define una relación de varios a varios entre una entidad y otra asociada.
  - *@ManyToOne*: Define una relación de varios a uno entre una entidad y otra asociada.
  - *@OneToMany*: Define una relación de uno a varios entre una entidad y otra asociada.
  - *@OneToOne*: Define una relación de uno a uno entre una entidad y otra asociada.
  - *@JoinColumn*: Se utiliza para indicar el nombre de la columna de la clave foránea en las relaciones de varios a uno y de uno a varios.

Clases Embeddable (1)

- La *annotation* @**Embeddable** permite definir una clase que no será mapeada a una tabla, pero se podrá utilizar como tipo de dato (compuesto) para utilizar en alguna propiedad de una entidad JPA.
- Las propiedades de la clase embeddable mapearán a columnas de la tabla de la entidad que la utiliza.
- En la clase de la entidad que tiene una propiedad del tipo de la clase embeddable, debe marcarse dicha propiedad con la annotation @Embedded.

Clases Embeddable (2)

```
@Embeddable
public class Direccion {
    private String calle;
    private int numero;
    ...
}

@Entity
public class Cliente {
    @Embedded
    private Direccion direccion;
    ...
}
```

Claves Primarias compuestas (1)

- Para mapear una clave primaria de más de una columna, debe crearse una clase adicional que la represente.
- Existen dos variantes:
  - La clase de la clave primaria compuesta debe ser ***@Embeddable*** y serializable. En la clase de la entidad debe incluirse una propiedad del tipo de la clase de la clave primaria compuesta y marcarla como ***@EmbeddedId***.
  - La clase de la clave primaria compuesta debe ser serializable. La clase de la entidad debe incluir la annotation ***@IdClass*** pasando por parámetro una instancia de **Class** de la clase de la clave primaria compuesta. Además debe repetir los atributos de la clave primaria y marcar cada uno de estos con ***@Id***.

Claves Primarias compuestas (2)

```
@Entity
public class MiEntidad {
    @EmbeddedId
    private MiClave miClave;
    ...
}
@Embeddable
public class MiClave implements Serializable {
    @Column(name="IdA", nullable="false")
    private int idA;
    @Column(name="IdB", nullable="false")
    private int idB;
    ...
}
```

Claves Primarias compuestas (3)

```
@Entity
@IdClass(MiClave.class)
public class MiEntidad {
    @Id
    private int idA;
    @Id
    private int idb;
    ...
}
public class MiClave implements Serializable {
    private int idA;
    private int idB;
    ...
}
```

Estrategias de Herencia (1)

- **SINGLE_TABLE**
  - Mapea a una única tabla que contiene los datos de toda la jerarquía (generalización y especializaciones), más una columna para indicar el discriminador de especialización.
  - Las columnas que para una fila no apliquen, se completan con el valor NULL.
  - En la clase base se utilizan las siguientes *annotations*:

    @Inheritance(strategy = nheritanceType.SINGLE_TABLE)

    @DiscriminatorColumn(name = "tipo")
  - En las clases derivadas se utiliza la *annotation*:

    @DiscriminatorValue(value="T1") // o "T2", ...

Estrategias de Herencia (2)

- **JOINED**
  - Mapea a una tabla para la generalización y una tabla para cada una de las especializaciones, utilizando el id de entidad como clave foránea en las especializaciones para relacionarlas con la generalización.
  - En la clase base se utiliza la siguiente *annotation*:

```
@Inheritance(strategy = InheritanceType.JOINED)
```

  - En las clases derivadas se utiliza la *annotation*:

```
@PrimaryKeyJoinColumn(name="IdEstaTabla",
                      referencedColumnName="IdTablaReferenciada")
```

  - Si se utiliza un id de entidad compuesto, puede indicarse así:

```
@PrimaryKeyJoinColumns({@PrimaryKeyJoinColumn
   (name="Id1",referencedColumnName="Id1"),
@PrimaryKeyJoinColumn(name="Id2",referencedColumnName="Id2")})
```

Estrategias de Herencia (3)

- **TABLE_PER_CLASS**
  - Mapea únicamente a una tabla para cada una de las especializaciones.
  - En la clase base se utiliza las siguientes *annotations*:

```
@Inheritance(strategy =
    InheritanceType.TABLE_PER_CLASS)
```

  - En las clases derivadas no es necesario agregar ninguna *annotation* adicional para la estrategia de herencia.

Unidad de Persistencia (1)

- Una unidad de persistencia es una pieza fundamental de configuración, que contiene la información necesaria para que la fábrica de gestores de entidades cree estos objetos que nos permitirán operar sobre los datos.
- Se define(n) en un archivo xml (típicamente con el nombre persistence.xml).
- Cada unidad de persistencia definida en este archivo incluye (entre otras cosas):
  - El nombre de la unidad de persistencia (para indicárselo a la fábrica de gestores de entidades).
  - Cuáles serán las clases de entidades manejadas por dicha unidad de persistencia.

Unidad de Persistencia (2)

- Cada unidad de persistencia definida en este archivo incluye (entre otras cosas):
  - El proveedor JPA de acceso a datos a utilizar en tiempo de ejecución.
  - Los datos de la conexión con la base de datos (driver, url de conexión, usuario, contraseña, etc.).
  - La estrategia de creación / validación del esquema de la base de datos.

Gestor de Entidades (1)

- Un gestor de entidades (EntityManager) es un objeto que permite realizar las operaciones de persistencia con las entidades JPA.
- Para crearlo debe utilizarse una fábrica (EntityManagerFactory), invocando a su método createEntityManager().
- La fábrica de gestores de entidades puede crearse invocando el método estático createEntityManagerFactory(String) de la clase Persistence. El String que recibe por parámetro es el nombre de la unidad de persistencia a utilizar.

Gestor de Entidades (2)

- El gestor de entidades (**EntityManager**) contiene una transacción de entidades (**EntityTransaction**) que se puede obtener con su método **getTransaction**().
- Toda vez que se necesite hacer algún cambio sobre los datos debe iniciarse una transacción con el método begin() del objeto **EntityTransaction**, y finalizarla con **commit**() en caso de éxito, o con **rollback**() en caso de producirse algún error.
- Asimismo, el gestor de entidades (**EntityManager**) permite crear objetos **Query** para realizar consultas sobre los datos. Este objeto se crea con el método **createQuery(String**) del gestor de entidades. El String que recibe por parámetro es la consulta en sintaxis JPQL.

Operaciones con los datos (1)

- Para dar de alta una entidad:
  1. Crear un gestor de entidades.
  2. Obtener su transacción e iniciarla.
  3. Invocar al método **persist(Object**) del gestor de entidades pasando por parámetro la entidad a persistir.
  4. Finalizar la transacción.

Operaciones con los datos (2)

- Para modificar una entidad:
  1. Crear un gestor de entidades.
  2. Obtener la entidad a modificar invocando al método **find(Class, Object)** sobre el gestor de entidades.
     1. El primer parámetro es una instancia de **Class** correspondiente al tipo de entidad a buscar.
     2. El segundo parámetro es el valor de la clave primaria (en caso de ser compuesta debe utilizarse una instancia de la clase de la clave primaria).
  3. Obtener la transacción del gestor de entidades e iniciarla.
  4. Realizar las modificaciones correspondientes sobre la entidad con sus métodos **setX**().
  5. Finalizar la transacción.

Operaciones con los datos (3)

- Para eliminar una entidad:
  1. Crear un gestor de entidades.
  2. Obtener la entidad a eliminar invocando al método find(Class, Object) sobre el gestor de entidades.
  3. Obtener la transacción del gestor de entidades e iniciarla.
  4. Invocar al método remove(Object) sobre el gestor de entidades pasándo por parámetro la entidad a eliminar.
  5. Finalizar la transacción.

Operaciones con los datos (4)

- Para consultar datos:
  1. Crear un gestor de entidades.
  2. Crear un objeto Query con la consulta JPQL.
  3. Ejecutar la consulta y obtener su resultado invocando sobre el objeto Query a alguno de los siguientes métodos (dependiendo de la cantidad de elementos que ésta devuelva):

     getResultList() : List

     getSingleResult() : Object

Operaciones con los datos (5)

- Al utilizar el método **find(Class, Object**) nos aseguramos que el gestor de entidades se encuentra gestionando la entidad devuelta.
- En caso de querer modificar o eliminar una entidad obtenida por fuera del gestor de entidades actual, dicha entidad debe en primer lugar registrase frente al gestor de entidades con el método **merge(Object**).