Frameworks de persistencia - JPA

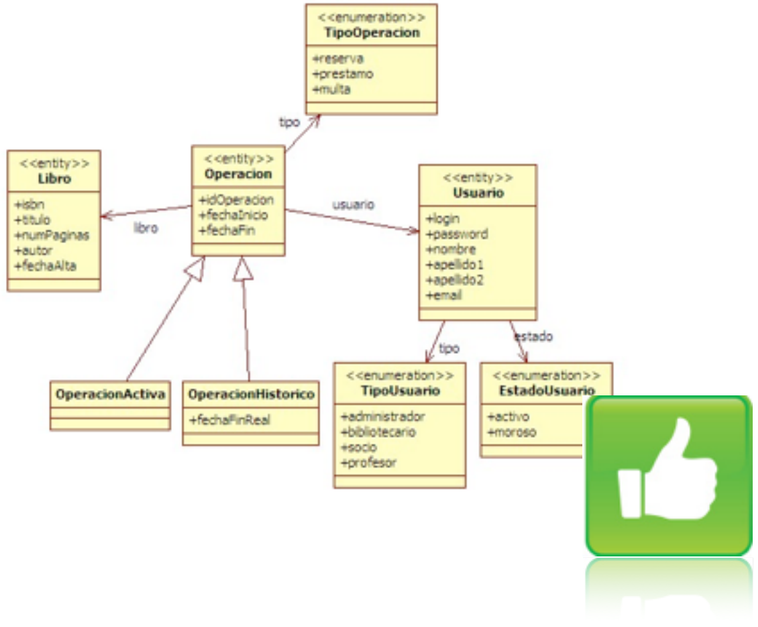
Sesión 1 - Primeros pasos con JPA

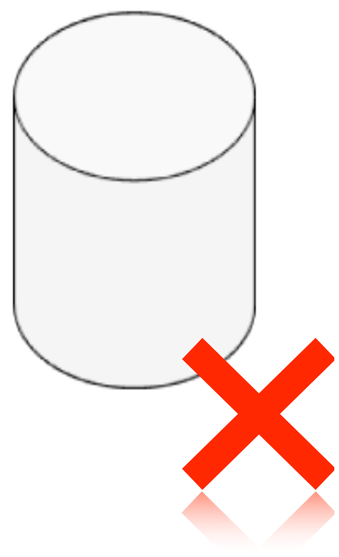
Índice

- Historia de JPA
- Conceptos básicos de JPA
- Demostración paso a paso

Java Persistence API

- Mediante JPA es posible trabajar directamente en Java con clases y objetos persistentes.
- ¡Ya no necesitaremos más SQL, ni JDBC¡





Java Persistence API

- Uso de anotaciones para especificar propiedades
- Entidades persistentes y relaciones entre entidades
- Mapeado objeto-relacional
- Gestión de contextos de persistencia y de transacciones
- Lenguaje de queries
- Empezaremos viendo JPA gestionado por la aplicación (JPA con Java SE) y después gestionada por el contenedor web / servidor de aplicaciones (JPA con Java EE)

Historia de JPA

- Hibernate nace en 2001, como un proyecto opensource dirigido por Gavin King
- En las mismas fechas se proponen las especificaciones oficiales de persistencia en Java EE
- Ninguna especificación oficial tiene éxito y Hibernate se convierte en el ORM más usado
- En Mayo de 2003 se constituye el grupo de trabajo que definirá la especificación de EJB 3.0 y Java EE 5 y Gavin King forma parte de él
- El grupo adopta el enfoque de Hibernate para las entidades persistentes y el estándar JPA 1.0 y Java EE 5 se aprueba en Abril de 2006
- JPA sigue formando parte del estándar Java EE 6 y 7
- Hibernate sigue mostrando una “marca” diferenciada de JPA, pero es la misma implementación con distintos nombres de clases (cuidado al incluir los imports: muchas veces coinciden los nombres de las clases y hay que incluir el package jpa.* en lugar del hibernate.* para trabajar con el estándar Java EE)

Java Persistence API

- Es un API Java estándar que se incluye en la última especificación Java EE 5 (en la JSR 220)
- Implementa un ORM (gestor de mapeado entidad-relación) basado en Hibernate
- Puede usarse en Java SE y en aplicaciones web sin servidor de aplicaciones
- Algunas características:
  - Basado en POJOs (Plain Old Java Objects, objetos sencillos Java frente a otros enfoques para hacer objetos persistentes que están directamente conectados con la base de datos mediante un complicado mecanismos de herencia e interfaces)
  - Basado en anotaciones
  - Simplicidad: abundantes opciones por defecto
- JPA 2.0 aprobada con Java EE 6 en diciembre de 2009
- La última versión 2.1 de JPA, aprobada con Java EE 7 en abril de 2013, se define en el JSR 338

Novedades de JPA 2.0

- Extensión de las opciones del mapeo objeto-relacional
- Soporte para colecciones de objetos embebidos
- Listas ordenadas
- Combinación de tipos de acceso
- API Criteria para la construcción de consultas
- Metadata adicional para la generación de DDL (Data Definition Language)
- Soporte para validación
- Soporte para cache de objetos compartidos

Novedades de JPA 2.1

- Conversores que permiten customizar el código de conversión entre los tipos de las clases java y los de la base de datos
- Actualizaciones y borrados en bloque mediante el API Criteria
- Consultas que ejecutan procedimientos almacenados en la base de datos
- Generación del esquema
- Grafos de entidades que permiten la recuperación o la mezcla parcial de objetos en memoria
- Mejoras en el lenguaje de consultas JPQL/Criteria: subconsultas aritméticas, funciones genéricas de base de datos, cláusula Join ON

Implementaciones de JPA

- Siguiendo el modo de funcionamiento habitual del mundo Java, una vez aprobado el estándar los distintos fabricantes sacan al mercado sus implementaciones
- Implementaciones de JPA:
  - Hibernate (JPA 2.1, usado por JBoss/WildFly - RedHat)
  - EclipseLink (JPA 2.1, usado por GlassFish - Oracle)
  - OpenJPA (JPA 2.0)

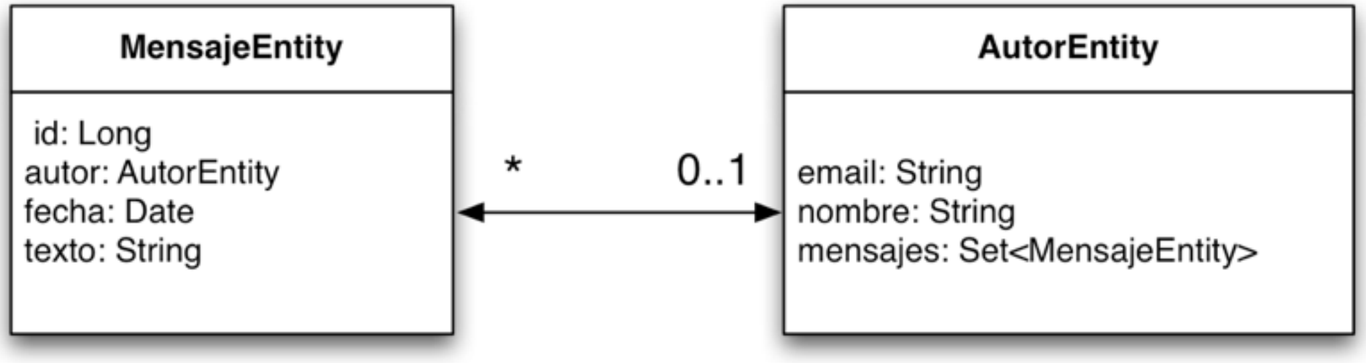
JPA se basa en JDBC

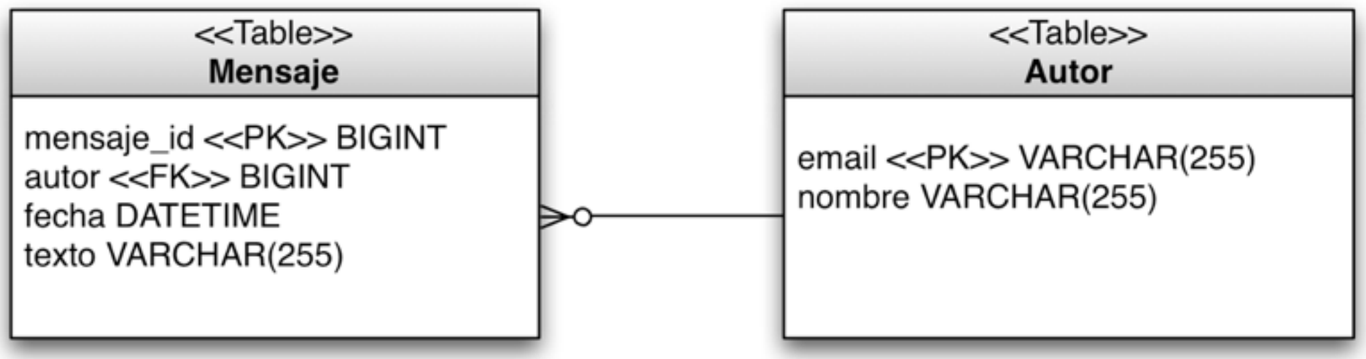
- La conexión de JPA con la base de datos se hace con un driver JDBC
- JPA abstrae los conceptos de bajo nivel: desaparecen los result sets y las consultas SQL

Mapeado entidad-relación

- El mapeado entidad-relación (ORM en inglés) es uno de los aspectos centrales de JPA
- Las entidades (clases Java) se mapean en tablas
- Los atributos de las clases en columnas
- Mapeos especiales para las relaciones y la herencia

Entidades y tablas





Entidad Autor

```java
@Entity
@Table(name="Autor")
public class Autor {
    @Id
    @GeneratedValue
    @Column(name = "autor_id")
    Long id;
    private String correo;
    private String nombre;
    @OneToMany(mappedBy = "autor", cascade = CascadeType.ALL)
    private Set<Mensaje> mensajes = new HashSet<Mensaje>()

    // constructor vacío
    // constructor con claves naturales
    // getters y setters
    // hashCode y equals con claves naturales
}
```

Entidad Mensaje

```java
@Entity
@Table(name="Mensaje")
public class MensajeEntity {

   @Id @GeneratedValue @Column(name="mensaje_id")
   private Long id;
   private String texto;
   private Date fecha;
   @ManyToOne
   private AutorEntity autor;
   ...
}
```

Entidades como POJO

- Las entidades son objetos Java normales que viven en el contexto de persistencia (memoria) del entity manager
- Estas entidades se crean y modifican como objetos Java normales
- El *entity manager* se encarga de generar las sentencias SQL que se envían a JDBC para mantener sincronizados el contexto de persistencia y la base de datos

Ejemplo básico

```
public class NuevoAutorMensaje {
   public static void main(String[] args) {
      Autor autor;

      EntityManagerFactory emf = Persistence.createEntityManagerFactory("mensajes");
      EntityManager em = emf.createEntityManager()
      em.getTransaction().begin();
      String email = leerTexto("Introduce el correo electrónico: ");
      String nombre = leerTexto("Introduce nombre: ");
      autor = new Autor(nombre, email);
      em.persist(autor);
      String mensajeStr = leerTexto("Introduce mensaje: ");
      Mensaje mens = new Mensaje(mensajeStr, autor);
      mens.setFecha(new Date());
      em.persist(mens);
      em.getTransaction().commit();
      em.close();
      emf.close();
}
```

EntityManager y transacciones

- Se puede pensar en un entity manager como en una conexión a la base de datos
- La creación de *entity managers* es barata, porque Hibernate mantiene un pool de objetos ya creados
- El *entity manager* mantiene su contexto de persistencia y "vigila" los objetos gestionados en él
- La forma de ligar una entidad nueva a un *entity manager* es mediante el método `persist()`. También podemos obtener una entidad a partir de datos de la base de datos llamando a su método `find()`.
- En JPA es obligatorio el uso de transacciones para actualizar los datos: las sentencias SQL que sincronizan la base de datos y el contexto de persistencia se realizan cuando se hace un commit de la transacción o cuando se realiza una consulta

El entity manager sincroniza las entidades

- Las entidades en el contexto de persistencia (memoria) se sincronizan con la BD mediante sentencias SQL que vuelca (*flush*) el *entity manager*
- La sincronización tiene lugar cuando se cierra la transacción con un `commit()` o cuando se realiza una consulta
- En JPA siempre es necesario abrir y cerrar una transacción, incluso para hacer una única modificación
- Al cerrar el *entity manager*:
  - No es posible realizar ninguna operación más con él
  - Las entidades quedan desconectadas (*detached*) de la base de datos

Factoría de entity manager y unidad de persistenca

- Los entity managers se crean a partir de una factoría que se obtiene con la instrucción `Persistence.createEntityManagerFactory("mensajes")`
- Es una llamada costosa, ya que debe JPA realiza en este momento el mapeo entre entidades y base de datos: hay que hacerlo sólo una vez al comenzar la aplicación
- El nombre "mensajes" hace referencia a una unidad de persistencia definida en el fichero de configuración `persistence.xml`

Fichero de configuración

- El fichero de configuración de la aplicación JPA se denomina `persistence.xml` y debe estar en el directorio `META-INF` en el classpath
- En él se declaran:
  - Clases entidad
  - Tipo de base de datos y driver de conexión
  - Parámetros de conexión (url, usuario y contraseña)
  - Parámetros que configuran la forma de configurar el esquema de datos (hibernate.hbm2ddl.auto) al conectarse a la base de datos

hibernate.hbm2ddl.auto

- El parámetro hibernate.hbm2ddl.auto dentro del persistence.xml define la forma de realizar el mapeado entre entidades y tablas de la base de datos
- Posibles valores
  - `update`: se actualiza el esquema de las tablas si ha habido algún cambio en las clases Java. Si no existen, se crean.
  - `validate`: se valida que el esquema se puede mapear correctamente con las clases Java. No se cambia nada de la base de datos.
  - `create`: se crea el esquema, destruyendo los datos previos.
  - `create-drop`: se crea el esquema y se elimina al final de la sesión.
- Es aconsejable utilizar el valor `create` en entornos de test, update en desarrollo y el `validate` en producción. Inicialmente, antes de ejecutar el primer programa, lo vamos a definir como update para probar a crear las tablas y comprobar cuál es el esquema de datos generado

Proyecto ejemplo: jpa-mensajes

- Dos entidades, Autor y Mensaje
- Una relación uno-a-muchos entre Autor y Mensaje
- Varios programas ejemplo:
  - Añadir autor y mensaje
  - Añadir mensaje
  - Buscar mensajes
- Haremos una introducción a los conceptos más importantes de JPA, mostrando cómo se implementan en la aplicación

Ejemplo práctico

- Demostración paso a paso: desarrollamos todos el ejemplo siguiendo los apuntes
- Veremos también en la demostración cómo realizar pruebas con JPA utilizando DbUnit

¿Preguntas?