Java Persistence API

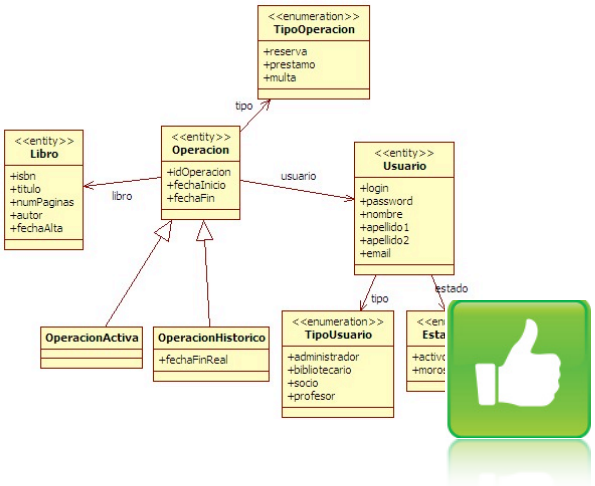
- Sesión 1: Introducción a JPA

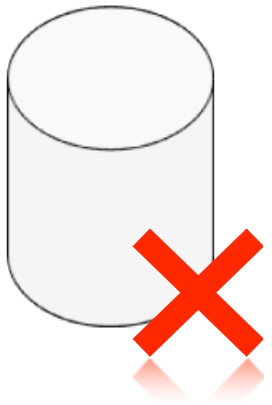
Índice

- Introducción al módulo JPA
- Historia de JPA
- Mapeado Entidad-Relación
- Entidades y contexto de persistencia
- Primera aplicación JPA

Java Persistence API

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





Módulo 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
- Veremos JPA gestionado por la aplicación (dejamos para el módulo de EJB el JPA gestionado por el contenedor)

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
  - Basado en anotaciones
  - Simplicidad: abundantes opciones por defecto

Historia de JPA

- Hibernate nace en 2001, como un proyecto opensource dirigido por Gavin King
- En las mismas fechas se proponen las especificaciones oficiales:
  - Entity Beans Java EE
  - JDO en Java SE
- 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 (la actual) y Gavin King forma parte de él
- El grupo adopta el enfoque de Hibernate para las entidades persistentes y el estándar se aprueba en Abril de 2006

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 (http://jpa.hibernate.org)
  - Apache OpenJPA (http://openjpa.apache.org/)
  - Oracle Toplink (http://www.oracle.com/technology/jpa)
  - CocoBase (http://www.thoughtinc.com/) - comercial

Aplicación ejemplo: HolaMundo.java

- Dos entidades, `Autor` y `Mensaje`
- Una relación uno-a-muchos entre `Autor` y `Mensaje`
- Un ejemplo sencillo:
  - 1. Buscamos un autor
  - 2. Si no existe lo creamos
  - 3. Creamos un mensaje nuevo de ese autor
  - 4. Obtenemos los mensajes del autor
- Haremos una introducción a los conceptos más importantes de JPA, mostrando cómo se implementan en la aplicación

Conceptos fundamentales de JPA

- Capa que se basa en JDBC
- Mapeado Entidad-Relación
- Entity Manager y contexto de persistencia
- Transacciones

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

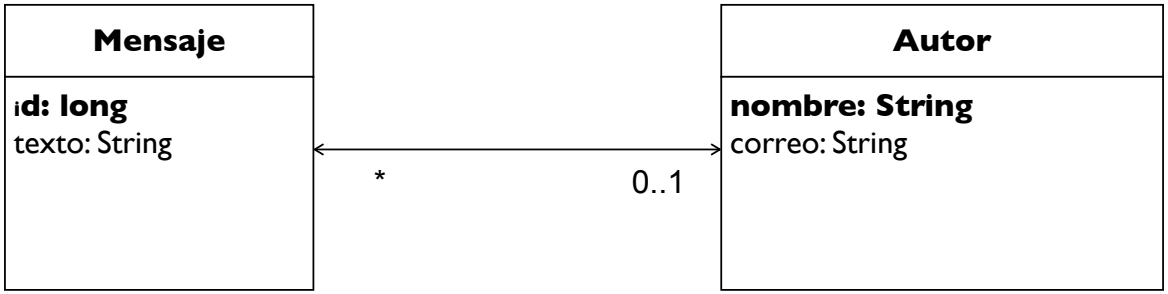
HolaMundo: Mensaje

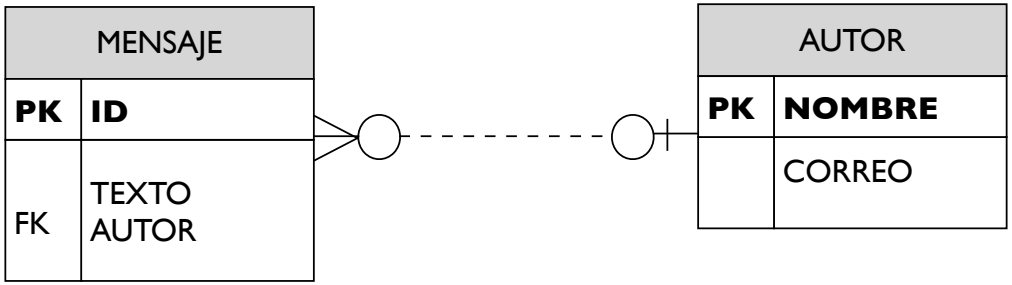
```
@Entity
public class Mensaje {

   @Id @GeneratedValue
   private long id;
   private String texto;
   @ManyToOne
   private Autor autor;

   private Mensaje() {}

   // getters y setters
}
```

HolaMundo: Autor

```
@Entity
public class Autor {
   @Id
   private String nombre;
   private String correo;
   @OneToMany(mappedBy = "autor")
   private Set<Mensaje> mensajes = new HashSet<Mensaje>();

   private Autor() {
   }

   // ...getters y setters

   public Set<Mensaje> getMensajes() {
      return mensajes;
   }

   public void setMensajes(Set<Mensajes> mensajes) {
      this.mensajes = mensajes;
   }
```

Mapeado entidad-relación





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

EntityManager y transacciones

- 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

HolaMundo: EntityManager, transacciones

```
// (1) Creamos el Entity Manager
EntityManager em = emf.createEntityManager();

   // (2) Creamos la transacción
   em.getTransaction().begin();

      // (3) Operaciones sobre las entidades
      autor = em.find(Autor.class, autorStr);
      if (autor == null) {
         autor = new Autor(autorStr, autorStr + "@ua.es");
         em.persist(autor);
      }
      mensaje = new Mensaje(mensStr, autor);
      em.persist(mensaje);
      mensaje.setAutor(autor);
      autor.addMensaje(mensaje);
      Collection<Mensaje> mensajes = autor.getMensajes();

   // (4) Cerramos la transaccion
   em.getTransaction().commit();

// (5) Cerramos el Entity Manager
em.close();
```

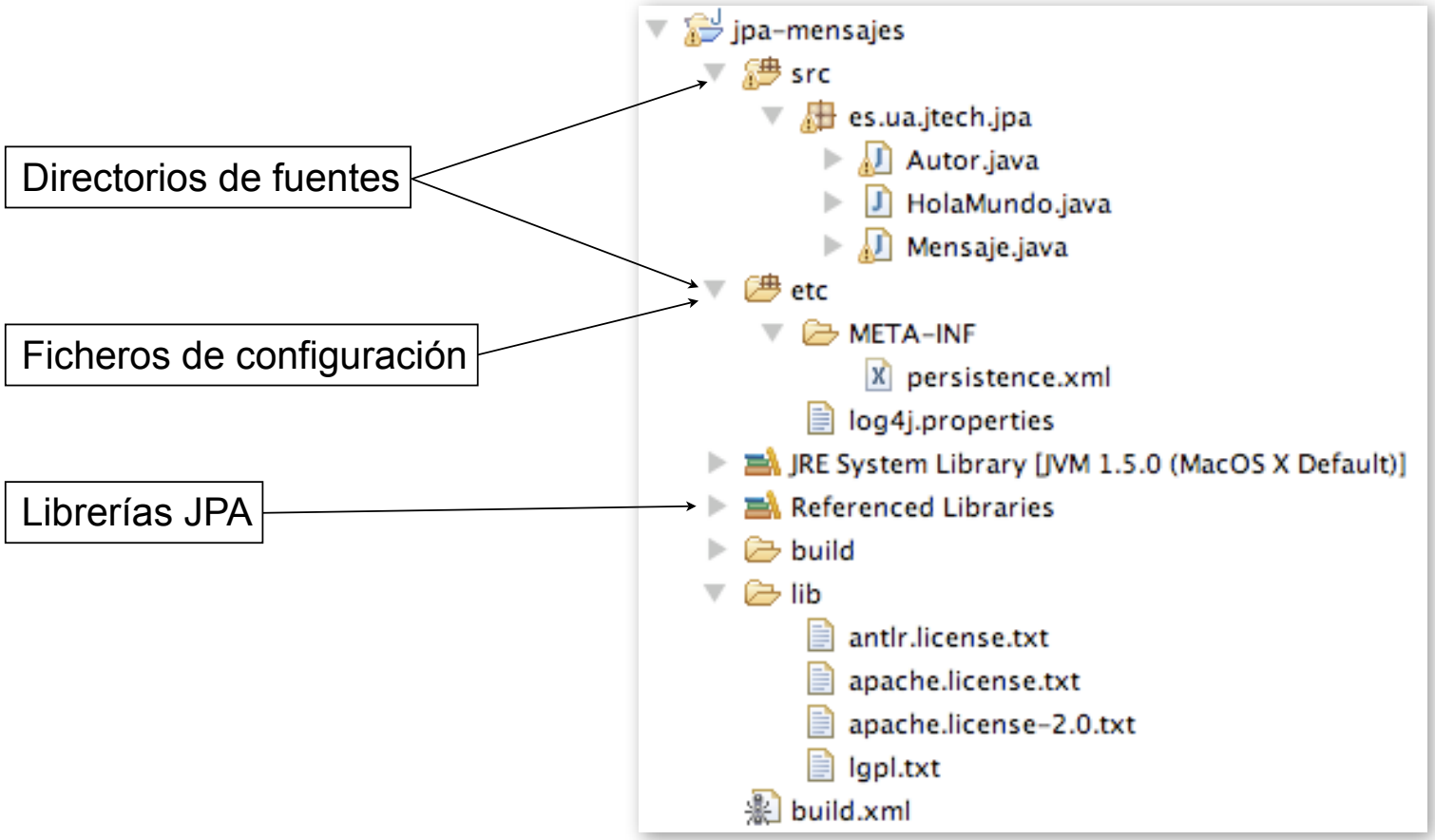
¿Cuando se sincroniza la BD?

- 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

¿Qué pasa cuando se cierra el entity manager?

- No es posible realizar ninguna operación más con él
- Las entidades quedan desconectadas (*detached*) de la base de datos
- Todavía no se hace un *flush*, puede ser que haya una transacción abierta

Configuración de Eclipse



Aplicación web

- El funcionamiento es similar a la aplicación Java
- En cada llamada a un servlet (petición HTTP) realizamos el ciclo de trabajo visto:
  - Se crea el entity manager
  - Se abre la transacción
  - Se realizan las operaciones con las entidades
  - Se cierra la transacción
  - Se cierra el entity manager

Código fuente aplicación web

```
public class AddMensaje extends HttpServlet {
   private static final long serialVersionUID = 1L;

   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
   throws ServletException, IOException   {
      String mensStr =
         request.getParameter("mensaje");
      String autorStr =
         request.getParameter("autor");

      EntityManagerFactory emf = Persistence
                           .createEntityManagerFactory("simplejpa");
      EntityManager em = emf.createEntityManager();

      //... operaciones con las entidades

      request.setAttribute("autor", autor);
      request.setAttribute("mensajes", mensajes);

      //... mostramos los resultados
      getServletContext().getRequestDispatcher("/listaMensajes.jsp")
                         .forward(request, response);

      em.close();
      emf.close();
   }
}
```

¿Preguntas?