

# JPA – Persistencia de entidades

Universidad Centroamericana “José Simeón Cañas”  
Ciclo 01-2020

# Antes de comenzar, consideremos lo siguiente del formulario de esta clase:

- El controlador encargado de mostrar la información del cliente posterior a realizar la búsqueda recibirá el objeto Cliente como un ModelAttribute, por lo que utilizaremos la propiedad **th:object** del formulario:

```
<form method="post" th:action="@{/guardar15}" th:object="${cliente}"> 1
  <input type="hidden" name="ccliente" class="form-control" th:field="*{ccliente}" /> 2
  <table cellpadding="10">
    <tr>
      <td><input type="button" class="btn btn-secondary" value="Regresar" onclick="location.href='./index15'" /></td>
    </tr>
    <tr>
      <td><label>Nombres: </label></td> 3
      <td><input type="text" id="snombres" class="form-control" th:field="*{snombres}"></td>
    </tr>
  </table>
</form>
```

1. Utilizamos la propiedad **th:object**, el cual recibe el nombre del objeto que viene en el ModelAndView que manda a llamar a esta página (método **buscar15** en la clase ClienteController.java)
2. Utilizamos un input **hidden** el cual estará mapeado a la propiedad **ccliente**, que contendrá el valor de la llave primaria del cliente, el objetivo de poner la llave primaria como un hidden, es para que viaje en el objeto al momento de realizar la petición al controlador **guardar15**, y sepamos que es una entidad existente (al tener llave primaria)
3. Seteamos cada una de las propiedades del objeto a cada input con la propiedad **th:field**

# Se utiliza para persistir una entidad los métodos **persist** y **merge** del **entityManager**

- Si es una entidad nueva (insert) la que se persistirá se utilizará el método **persist(T Entidad)** que recibe de parámetro el objeto entidad que queremos guardar
- Si es una entidad existente (update) la que se persistirá entonces se utiliza el método **merge(T Entidad)** que recibe de igual forma el objeto entidad que queremos actualizar

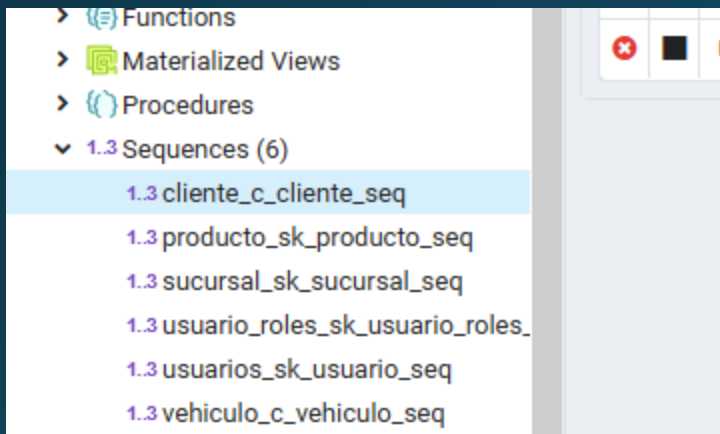
Se creará el método **saveCliente(Cliente)** en el DAO de Cliente, que persistirá el registro de cliente que se le envía desde la JSP como un `ModelAttribute`

# Debemos primero configurar la secuencia en la propiedad clave del dominio

- Hibernate y JPA pueden utilizar automáticamente el valor de una secuencia si una columna es definida como tal.
- En nuestro caso para la tabla Cliente, la columna **c\_cliente** la hemos definido como **serial** (autoincrementable)
- Debido a esto, debemos configurar debidamente la propiedad **ccliente** de nuestra clase dominio para que utilice la secuencia de la columna **serial**

# Secuencia en PgAdmin4

- La secuencia que utilizaremos la podemos ver en el PgAdmin4 en el apartado **sequences** del esquema **store**



Como mencionábamos anteriormente, dicha secuencia fue creada automáticamente por PostgreSQL al crear la tabla con la columna **c\_cliente** con tipo de dato **serial**

# Configuración de la secuencia en la propiedad clave

```
@Id
@GeneratedValue(generator="cliente_c_cliente_seq", strategy = GenerationType.AUTO)
@SequenceGenerator(name = "cliente_c_cliente_seq", sequenceName = "store.cliente_c_cliente_seq")
@Column(name = "c_cliente")
private Integer ccliente;
```

**@SequenceGenerator:** Se define la secuencia a la que estará haciendo referencia la anotación **GeneratedValue**. La propiedad **name** define el nombre con la que se referenciará esta secuencia (es la que también se define en la propiedad **generator** de la anotación **GeneratedValue**, y la propiedad **sequenceName** define el nombre de la secuencia en la base de datos (esquema incluido)

**@GeneratedValue:** Se utiliza para anotar a la propiedad que actúa como llave primaria (por ende la que está anotada con **@Id**). La propiedad **generator** define el nombre del generador que se utilizará para insertar valores secuenciales a la propiedad. **Strategy** define la estrategia con la que se generará el valor, en nuestro caso lo dejaremos en **AUTO**.

# Crearemos el paquete **service**

- Recordemos que ahora, todas las interacciones con el objeto de Acceso a datos (DAO) se hará a través de la capa de servicio, mediante la creación de una interfaz y la clase implementadora de la entidad correspondiente.
- Crearemos el paquete **com.uca.capas.modelo.service** y crearemos la interfaz ClienteService, donde definiremos los métodos de lectura y persistencia de la entidad Cliente

```
public interface ClienteService {  
    public List<Cliente> findAll() throws DataAccessException;  
    public Cliente findOne(Integer codigo) throws DataAccessException;  
    public void save(Cliente c) throws DataAccessException;  
}
```

Si vemos, le hemos puesto los mismos nombres de los métodos del DAO, sin embargo, son totalmente diferentes como veremos en su implementación

# Ahora crearemos la clase implementadora

- Llamaremos a la clase **ClienteServiceImpl.java** y tendrá lo siguiente:

```
@Service
public class ClienteServiceImpl implements ClienteService {

    @Autowired
    ClienteDAO clienteDao;

    public List<Cliente> findAll() throws DataAccessException {
        return clienteDao.findAll();
    }

    public Cliente findOne(Integer codigo) throws DataAccessException {
        return clienteDao.findOne(codigo);
    }

    @Transactional
    public void save(Cliente c) throws DataAccessException {
        clienteDao.save(c);
    }
}
```

- **@Service:** Esta anotación le dicta a Spring que dicha clase es de Servicio, con lo cual se podrá manejar los métodos dentro de transacciones
- Ahora, en vez de inyectar el DAO en el controlador, lo haremos en esta clase, con la anotación **@Autowired**
- La lógica de los métodos es sencilla en este caso, ya que solo mandaremos a llamar a los métodos del DAO y retornaremos lo que nos devuelva
- El método **save** está siendo anotado por **@Transactional**, esto debido a que como es una operación de persistencia la que se realizará (INSERT o UPDATE) Spring necesita realizarlo dentro de una Transacción, caso contrario, nos lanzará una excepción mencionando que no hay Transacción en curso.
- Los métodos **findAll** y **findOne**, al no realizar operaciones de persistencia (solo de consulta) no precisan llevar esta anotación.



# Ahora desarrollaremos el método encargado de persistir la entidad

```
public void save(Cliente c) throws DataAccessException {  
    if(c.getCcliente() == null) { //Si la propiedad de la llave primaria viene vacío, entonces es un INSERT  
        entityManager.persist(c); //Utilizamos persist ya que es un INSERT  
    }  
    else { //Caso contrario, se busca al cliente, por lo que la propiedad ccliente viene llena (el input hidden del formulario)  
        entityManager.merge(c); //Utilizamos merge ya que es un UPDATE  
    }  
}
```

Para saber si es un INSERT o un UPDATE lo que realizaremos de la entidad, revisaremos la propiedad **Ccliente** (llave primaria), si viene llena, significa que primero se realizó una búsqueda del cliente y luego se guardaron los cambios.

Si no viene llena, significa que el usuario le dio clic al botón **Nuevo Cliente** (como veremos en el ejemplo mas adelante) y se envió un nuevo objeto Cliente al formulario correspondiente, y como dicha propiedad es manejada como un **hidden**, no es ingresada por el usuario, por lo que al hacer el submit del formulario irá vacío, dándonos la pauta que es un nuevo Cliente.

# Configuración de Transacciones en Spring

Para utilizar la anotación `@Transactional` debemos de configurar la funcionalidad en la clase de configuración de JPA

```
@Bean
JpaTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory);
    return transactionManager;
}
```

```
@Configuration
@EnableTransactionManagement
public class JpaConfiguration {
```

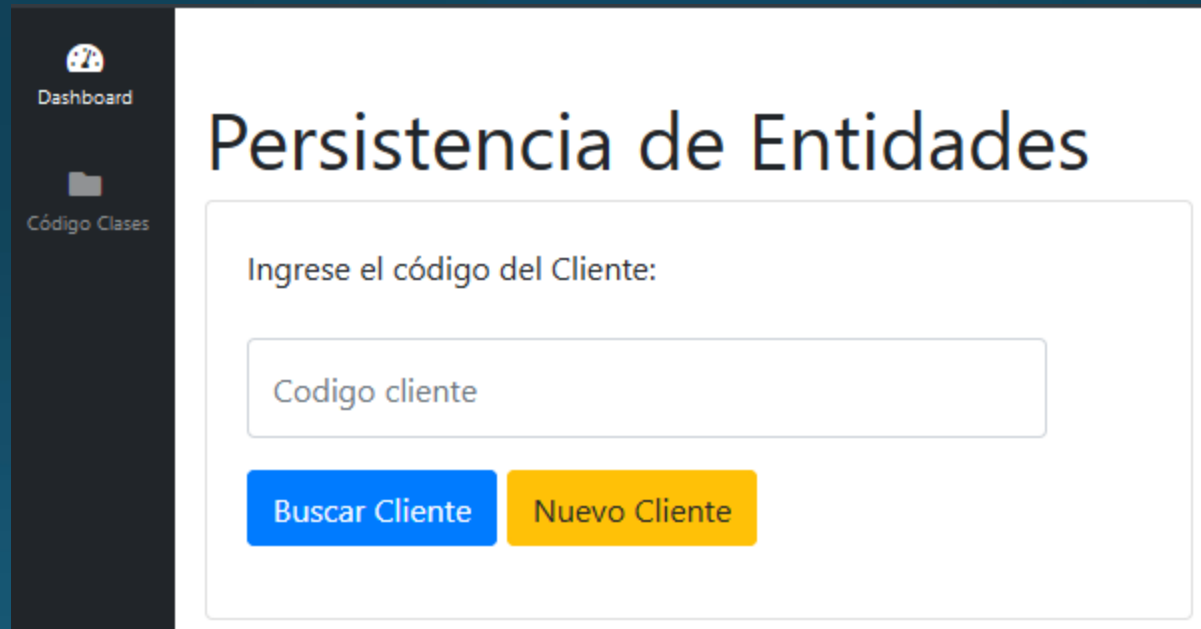
Configuramos un método que devuelva un objeto de tipo `JpaTransactionManager` y seteamos el `EntityManagerFactory` devuelto por el método correspondiente.

Por último anotamos la clase con `@EnableTransactionManagement`, para habilitar el soporte de Transacciones por Spring

Ejemplo

# Actualización de Entidades

- Ingresamos a la URL “/index15” o a través del Menú de la izquierda en Clase 15

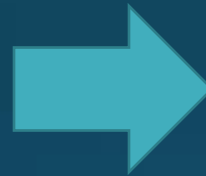


The screenshot shows a web application interface. On the left is a dark sidebar with two menu items: 'Dashboard' with a clock icon and 'Código Clases' with a folder icon. The main content area has a white background and is titled 'Persistencia de Entidades'. Below the title, there is a text prompt 'Ingresa el código del Cliente:' followed by a text input field containing the placeholder text 'Codigo cliente'. At the bottom of the form, there are two buttons: a blue button labeled 'Buscar Cliente' and a yellow button labeled 'Nuevo Cliente'.

# Ingresamos un código de Cliente

## Persistencia de Entidades

Ingrese el código del Cliente:



### Información del Cliente

Nombres:

Apellidos:

Fecha de Nacimiento:

Estado: ☒ Activo

Damos clic al botón "Buscar Cliente"

# Dar clic al botón “Guardar Cambios”

- Al dar clic al botón guardar y revisamos el objeto en modo de depuración en la línea 73 de ClienteDAOImpl.java, veremos lo siguiente:

```
public void save(Cliente c) throws DataAccessException {  
    if(c.getCcliente() == null) { //Si la propiedad de la llave p  
    }  
    else  
    {  
        > c = Cliente (id=142)  
        > bactivo= Boolean (id=146)  
        > ccliente= Integer (id=150)  
        > fnacimiento= Date (id=153)  
        > sapellidos= "Bril" (id=156)  
    }  
}
```

Efectivamente, como primero buscamos al cliente, la propiedad ccliente viene con el valor del cliente correspondiente, por lo que la condición lo enviará a ejecutar el método **merge** de la entidad.

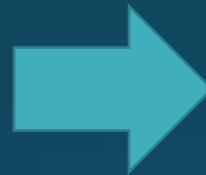
```
Hibernate: select cliente0_.c_cliente as c_cliente1_0_0_, cliente0_.b_activo as b_activo2_0_0_, cliente0_.f_nac  
Hibernate: select cliente0_.c_cliente as c_cliente1_0_0_, cliente0_.b_activo as b_activo2_0_0_, cliente0_.f_nac  
Hibernate: update store.cliente set b_activo=?, f_nacimiento=?, s_apellidos=?, s_nombres=? where c_cliente=?
```

Al revisar la consola posterior a realizar la operación de **merge**, vemos que efectivamente se ejecuta la sentencia **UPDATE** correspondiente en la base de datos

# Inserción de un nuevo Cliente

## Persistencia de Entidades

Ingresa el código del Cliente:



## Información del Cliente

Nombres:

Apellidos:

Fecha de Nacimiento:

Estado: ☐ Activo

Daremos clic al botón “Nuevo Cliente”, el cual, si vemos el controlador “/nuevocliente” en ClienteController.java, crea un ModelAndView, le setea un nuevo objeto Cliente, y lo redirige a la página cliente.html

# Ingresamos la información del Cliente

Información del Cliente

Regresar

Nombres:

Lorem

Apellidos:

Impsum

Fecha de Nacimiento:

05/08/1993

Estado:

☒ Activo

Guardar cambios

Luego damos clic al botón “Guardar Cambios”

```
public void save(Cliente c) throws DataAccessException {  
    if(c.getCcliente() == null) { //Si la propiedad de la l  
    }  
    else  
    }  
}
```

▼ c= Cliente (id=149)

- > bactivo= Boolean (id=153)
- > ccliente= null
- > fnacimiento= Date (id=157)
- > sapellidos= "Impsum" (id=161)

null

Ahora, al depurar el programa en el método save y examinamos el objeto Cliente, vemos que la propiedad ccliente viene **null**, debido a que no fue mediante un cliente existente que se mandó la petición, sino a través del nuevo objeto Cliente enviado al formulario por el controlador, por ende, dicha propiedad vendrá nula (ya que no es llenada por el usuario)



# Al finalizar el proceso de guardado

Si vemos la consola, aparecerá que se ejecutaron dos instrucciones:

```
2020-05-11 14:46:11.244 INFO 3296 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet DispatcherServlet
2020-05-11 14:46:11.244 INFO 3296 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 5 ms
Hibernate: select nextval ('store.cliente_c_cliente_seq')
Hibernate: insert into store.cliente (b_activo, f_nacimiento, s_apellidos, s_nombres, c_cliente) values (?, ?, ?, ?, ?)
```

La primera instrucción es para obtener el valor de la secuencia que le definimos en la entidad Cliente, con las anotaciones @SequenceGenerator y @GeneratedValue, dicho valor lo utilizará para poblar la columna c\_cliente.

La segunda instrucción es el INSERT correspondiente a la tabla **cliente**, como vemos, está parametrizada (con los signos de interrogación), dichos parámetros serán poblados con los valores que vengan del objeto Cliente.