

# Implementación de DAO con Hibernate

Universidad Centroamericana “José Simeón Cañas”  
Ciclo 01-2020

# Data Access Object

- Será el objeto encargado de realizar las operaciones con la base de datos
- Dicho objeto será utilizado desde el controlador
- Primero se define una interfaz con los métodos que tendrá nuestra clase de acceso a datos
- Luego creamos una clase implementando dicha interfaz, desarrollando cada uno de los métodos definidos en el

# Trabajaremos con la tabla “Cliente”

- La tabla se encontrará en el esquema “store” de la base de datos y tendrá las siguientes características:

```
CREATE TABLE store.cliente
(
  c_cliente serial NOT NULL,
  s_nombres character varying(30),
  s_apellidos character varying(30),
  f_nacimiento date,
  b_activo boolean,
  CONSTRAINT cliente_pkey PRIMARY KEY (c_cliente)
)
```

# Creación de la clase dominio "Cliente"

- Procederemos ahora a crear la clase que representará a dicha tabla "Cliente"
- Crearemos el paquete **com.uca.capas.domain**
- Luego vamos a crear la clase **Cliente.java**, que tendrá propiedades que estarán representando a cada columna de la tabla según el tipo de dato. Por ejemplo, si en la tabla la columna es de tipo **Int** o **Number**, la propiedad en la clase será **Integer**, si es **VARCHAR** entonces será **String**, y así correspondientemente para los demás tipos de datos.

```
@Entity
@Table(schema = "store", name = "cliente")
public class Cliente {

    @Id
    @Column(name = "c_cliente")
    private Integer ccliente;

    @Column(name = "s_nombres")
    private String snombres;

    @Column(name = "s_apellidos")
    private String sapellidos;

    @Column(name = "f_nacimiento")
    private Calendar fnacimiento;

    @Column(name = "b_activo")
    private Boolean bactivo;
```

**@Entity:** Anotación utilizada para especificar que esta clase hace referencia a una Entidad, al iniciar el aplicativo Spring escaneará las clases contenidas en el paquete definido en el método **setPackagesToScan** en la clase **JpaConfiguration**, al encontrar esta anotación manejará dichas clases como Entidades que harán referencia a una tabla.

**@Table:** Especifica la tabla a la que hace referencia la Entidad. Recibe dos parámetros, **schema** y **name**, en el cual se le define el esquema y el nombre de la tabla de la base de datos a la que hace referencia.

**@Id:** Especifica que esta propiedad es la llave primaria de la Entidad (y de la tabla)

**@Column:** Especifica la columna de la tabla a la que hace referencia la propiedad, recibe como propiedad el nombre de la columna (propiedad **name**)

# Creación de la interfaz de acceso a datos (DAO)

- Crearemos el paquete **com.uca.capas.modelo.dao** y crearemos la interfaz **ClienteDAO** con dos métodos:

```
public interface ClienteDAO {  
    public List<Cliente> findAll() throws DataAccessException;  
    public Cliente findOne(Integer codigo) throws DataAccessException;  
}
```

- `findAll()` devolverá una Lista de Clientes que contendrá a todos los clientes que se encuentren en la tabla Cliente, como objetos de tipo Cliente
- `findOne` es un método que devolverá un objeto de tipo Cliente, el cual recibirá de parámetro el valor de la llave primaria (un integer)

# Crearemos la implementación de la interfaz ClienteDAO

```
/*
 * Esta anotacion le dice a Spring que este es un objeto DAO, por lo que sera
 * manejado automaticamente y posteriormente podremos obtener una instancia
 * de este objeto mediante inyeccion de depdencias (@Autowired)
 */
@Repository
public class ClienteDAOImpl implements ClienteDAO {

    /*
     * Definimos el objeto EntityManager con el cual ejecutaremos
     * consultas a la base de datos, para esto utilizamos la anotacion
     * @PersistenceContext, al cual le definimos el nombre de la unidad
     * de persistencia que le fue asignado en la clase JpaConfiguration (línea 21)
     * con la propiedad unitName, con esto tenemos el objeto EntityManager
     * de la base de datos definida en nuestra clase de configuracion de Jpa
     */
    @PersistenceContext(unitName = "modelo-persistence")
    EntityManager entityManager;
```

# ClienteDAOImpl.java

```
@Override
public List<Cliente> findAll() throws DataAccessException {
    //Creamos un objeto StringBuffer para definir la consulta a ejecutar
    StringBuffer sb = new StringBuffer();
    //Definimos la consulta con el metodo append
    sb.append("select * from store.cliente");
    /*
     * Declaramos un objeto de tipo javax.persistence.Query, el cual representa a la consulta
     * Dicho objeto no lo instanciamos, sino que le asignamos lo que devuelve el metodo
     * createNativeQuery del entityManager, el cual recibe dos parametros
     * 1. La consulta de tipo String
     * 2. La referencia de la clase a la que queremos mapear el resultado (Cliente)
     */
    Query query = entityManager.createNativeQuery(sb.toString(), Cliente.class);
    /*
     * Ejecutamos la consulta con el metodo getResultList() de nuestro objeto Query
     * el cual devolvera una lista del tipo definido anteriormente (Cliente.class)
     * y lo asignamos a una lista de tipo cliente
     */
    List<Cliente> res = query.getResultList();
    //Devolvemos la lista con la coleccion de Clientes
    return res;
}
```

# ClienteDAOImpl.java

```
public Cliente findOne(Integer codigo) throws DataAccessException {  
    /*  
     * Para obtener un cliente en base a su llave primaria nos auxiliaremos  
     * del metodo find del objeto EntityManager, el cual recibe de parametro la  
     * referencia de la clase sobre la cual queremos buscar la entidad, y como  
     * segundo parametros el valor de la llave primaria, el cual es enviado como  
     * parametro en el metodo. Dicho metodo devolvera el objeto Cliente encontrado  
     * para esa llave primaria, sino lo encuentra devolverá NULL  
     */  
    Cliente c = entityManager.find(Cliente.class, codigo);  
    return c;  
}
```

Con esto, hemos desarrollado la implementación definida en nuestra interfaz ClienteDAO



Debemos crear siempre una Interfaz y su clase Implementadora, ya que, cuando vayamos a utilizar el objeto DAO, lo haremos referenciando a la interfaz, puesto que Spring automáticamente buscará a la clase que implementa dicha interfaz y nos devolverá una instancia de dicho objeto (Inyección de dependencias)

# Ahora crearemos los controladores que utilizarán los DAO

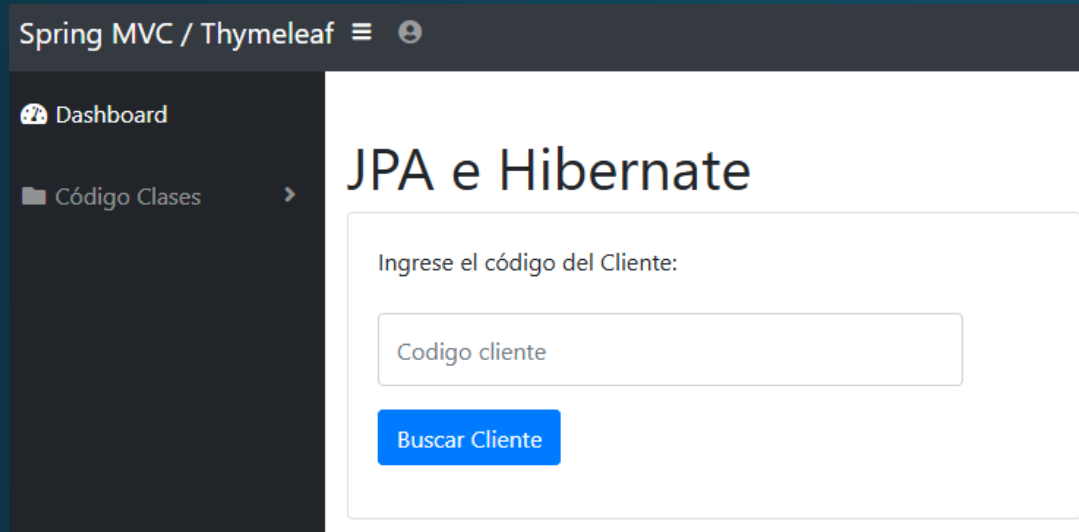
```
@Controller
public class ClienteController {

    @Autowired
    ClienteDAO clienteDao;

    @RequestMapping("/buscarcliente")
    public ModelAndView buscar(@RequestParam Integer codigo) {
        ModelAndView mav = new ModelAndView();
        Cliente c = clienteDao.findOne(codigo);
        mav.addObject("cliente", c);
        mav.setViewName("clases/clase13/cliente");
        return mav;
    }
}
```

**@Autowired:** Esta anotación se utiliza para inyectar el objeto al que estamos anotando. En este caso queremos inyectar el objeto de acceso a datos, por lo que anotamos **la interfaz** del DAO (no la clase implementadora, Spring automáticamente busca la implementación de la interfaz)

Al ejecutar el aplicativo e ingresar a la URL <http://localhost:8080/index13>, veremos la siguiente pantalla:



Al dar clic al botón “Buscar Cliente”, este hará una búsqueda sobre el código del cliente introducido como parámetro (sobre la llave primaria). Se enviará la petición al controlador con URL **/buscarcliente**

```
@RequestMapping("/buscarcliente")
public ModelAndView buscar(@RequestParam Integer codigo) {
    ModelAndView mav = new ModelAndView();
    Cliente c = clienteDao.findOne(codigo);
    mav.addObject("cliente", c);
    mav.setViewName("clases/clase13/cliente");
    return mav;
}
```

Se recibe la petición con el parámetro enviado por el formulario, y utilizaremos el método **findOne** de nuestro DAO, el cual recibe de parámetro el valor de la llave primaria del Cliente, y al encontrarlo, devolverá el registro correspondiente como un objeto de tipo **Cliente**, luego ingresamos el objeto en el ModelAndView y lo redirigimos a la página cliente.html, en el cual renderizaremos cada campo del objeto enviado.

# Métodos Delegate

- Son métodos que se encuentran en la clase domain, y que permiten devolver otro tipo de dato en función del valor de una propiedad
- Por ejemplo, si queremos devolver “Activo” o “Inactivo” según el valor del campo **b\_activo** que es booleano lo hacemos mediante un método delegate, ya que la propiedad devolverá **true** o **false**
- Entonces crearemos un método “delegador” o “delegate” que se encargará de devolver este valor

# Métodos Delegate

- Por ejemplo, si en el formulario HTML, a un campo le definimos una propiedad de la clase Domain que es de tipo Date o Calendar, lo veríamos de la siguiente manera:

Información del Cliente

Regresar

Nombres:

Di

Apellidos:

Maidlow

Fecha de Nacimiento:

java.util.GregorianCalendar[t

Estado:

☐ Activo

# Los métodos “delegate” son como cualquier método getter, pero con lógica por dentro

```
//Delegate para conversion de fecha
public String getFechaDelegate(){
    if(this.fnacimiento == null){
        return "";
    }
    else{
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        String shortdate = sdf.format(this.fnacimiento.getTime());
        return shortdate;
    }
}

//Delegate para activo o inactivo
public String getBactivoDelegate(){
    if(this.bactivo == null){
        return "";
    }
    else{
        if(this.bactivo) return "ACTIVO";
        else return "INACTIVO";
    }
}
```

# Para utilizarlos se hace directamente dentro de la página HTML

```
<tr>
  <td><label>Fecha de Nacimiento:</label></td>
  <td><input type="text" id="usuario" class="form-control" th:field="*{cliente.fechaDelegate}"></td>
</tr>
```

```
public String getFechaDelegate()
```



Se accede a el utilizando el mismo nombre del método delegate de la clase dominio, pero sin el prefijo "get" y la primera letra en minúscula independientemente si es mayúscula o no.

# Ahora, mostrará lo devuelto por dicho método

### Información del Cliente

Regresar

Nombres:

Di

Apellidos:

Maidlow

Fecha de Nacimiento:

04-04-1977

Estado:

☐ Activo

```
//Delegate para conversion de fecha
public String getFechaDelegate(){
    if(this.fnacimiento == null){
        return "";
    }
    else{
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        String shortdate = sdf.format(this.fnacimiento.getTime());
        return shortdate;
    }
}
```

Efectivamente, la fecha en formato dd-MM-yyyy